The Security of the Bitcoin Protocol

Saravanan Vijayakumaran
Associate Professor
Department of Electrical Engineering
Indian Institute of Technology Bombay

May 19, 2018

The Security of the Bitcoin Protocol

Saravanan Vijayakumaran

Version 0.1, May 2018

PDF template source: https://github.com/hmemcpy/milewski-ctfp-pdf.

Disclaimer

The views expressed in this document are my own and not of my employer, Indian Institute of Technology Bombay. The writing of this document has been paid for by Zeb IT Service Pvt Ltd, which operates under the popular brand name Zebpay.

Contents

Preface

In this paper, we consider the security of the Bitcoin protocol from an information technology perspective. Specifically, we attempt brief answers to the following questions:

- How difficult is it for an adversary to *steal another user's bitcoins?*

- How difficult is it for an adversary to *modify Bitcoin transactions*, both transactions already recorded on the blockchain and transactions waiting to be added to the blockchain?

- How difficult is it for an adversary to *disrupt the operation of the Bitcoin network*?

To set the context for the answers, we give a brief overview of the Bitcoin protocol design.

We do not address the security of Bitcoin from a financial perspective, i.e. we do not consider the question of whether Bitcoin is a secure investment. Bitcoin prices are determined by the market forces of supply and demand. The demand is a result of a loose global consensus regarding its current value and the potential for future value increases. While Bitcoin prices have risen in the past, there is no way to predict that this trend will continue or subside.

We focus on the security of the core Bitcoin protocol, which consists of a peer-to-peer network of nodes running open source Bitcoin clients. We do not consider the security of the exchanges, web wallets, and mobile applications which enable users to purchase, store, and trade bitcoins. While some of these applications are open source, most of them are based on proprietary code which is not available for review. Furthermore, the security of these systems is sometimes based on manual procedures for protecting the Bitcoin private keys, which are not documented in the public domain.

1

An Overview of Bitcoin

Bitcoin is a decentralized virtual currency. It exists as a peer-to-peer network of a few thousand nodes running software which implements the Bitcoin protocol. By the *Bitcoin protocol*, we mean the set of rules which the network nodes have to follow to continue active participation in the network. The protocol is specified by the implementation called *Bitcoin Core* whose source code is available at https://github.com/bitcoin/bitcoin/. Unlike traditional network protocols, there is no written specification of the Bitcoin protocol in plain English (or any other natural language). The Bitcoin Core implementation serves as the de facto specification. The first version of this implementation was released in 2009 by Satoshi Nakamoto, the pseudonymous creator of Bitcoin. While Nakamoto stopped contributing to Bitcoin Core development in 2010 and stopped all communication with the other developers soon after, the protocol has been continually refined and improved upon by hundreds of programmers. These programmers are referred to as *Core developers* in the Bitcoin community. They do not belong to any single company or institution and coordinate Bitcoin development through email discussions[1] in the bitcoin-dev mailing list, comments on new code submissions[2] to the Bitcoin Core GitHub repository, and virtual meetings[3] on Internet Relay Chat (IRC) channels.

Bitcoin is a decentralized system in two different ways:

- Firstly, the Bitcoin network does not rely on a central coordinator node for correct operation. Nodes in the network are free to leave at any time and the protocol operation is not affected by such individual exits.

- Secondly, the development and maintenance of the Bitcoin protocol is not reliant on a single person or company. The Bitcoin Core developers are a loose collection of individuals who work on Bitcoin for different reasons. Some of these developers work for companies which build applications on top of Bitcoin or sell goods/services related to Bitcoin. Other developers are either freelance consultants or hosted by academic institutions to work on Bitcoin. While individual developers in this community wield varying degrees of influence on the future direction of the Bitcoin protocol, community consensus is required to approve any changes.

  Furthermore, if any of the current Core developers were to abandon Bitcoin protocol development then other developers will step in to fill the void. These developers will be motivated by the opportunity to contribute to an immensely successful open source project.

[1] https://lists.linuxfoundation.org/mailman/listinfo/bitcoin-dev

[2] https://github.com/bitcoin/bitcoin/pulls

[3] https://bitcoincore.org/en/meetings/

1.1 Challenges in Building a Decentralized Virtual Currency

1.1.1 Determining Ownership

Ownership of units of fiat currency (rupees, dollars, yen) depends on their form. If these units are in the form of paper notes or metal coins, ownership is determined by physical possession. If the fiat currency units are represented by an entry in a bank account, then their ownership is determined by the possession of credentials required to spend them (online banking password, written signature on a cheque).

The units of a virtual currency like Bitcoin do not have a form analogous to fiat currency notes or coins. Ownership can only be determined by the possession of spending credentials in digital form. Credential validation in a *decentralized* virtual currency presents the following challenges:

- *Who maintains the database of valid credentials?*

  Even if the credentials are protected by a one-way function, a single node or a group of nodes cannot be given this responsibility in a network where nodes are free to leave at any time. One solution is to require all the nodes in the network to maintain the credentials database. This solution raises the question of how the validation of single credential will be performed. It is not feasible to query all the nodes in the network. If we restrict the query set to a small subset of the network nodes (say a few dozens), then a malicious party who controls this subset can validate invalid credentials.

- *How can one guarantee the integrity of a credentials database?*

  Integrity of digital data is typically ensured via digital signatures. But this assumes that we trust the signing party. In a decentralized network, it is risky to trust any one node which may later leave or become malicious.

These challenges are due to the lack of a *single source of trust* in decentralized systems. The absence of a party which all nodes in the network can trust is what makes building decentralized systems difficult.

1.1.2 Preventing Double Spending

Double spending refers to the problem of the owner of some units of a virtual currency spending it more than once. It is a problem unique to decentralized virtual currencies. When the owner of fiat currency note or coin spends it, he hands it off to the party receiving the payment and there is no question of double spending it. When fiat currency in a bank account is spent via an online transfer or cheque, the balance in the bank account is reduced immediately to reflect the spending. Subsequent payments from the bank account only spend the remaining balance and it is not possible to spend an amount more than once. In this context, the bank database is the *single source of truth* with regard to the amount of fiat currency owned by the account holder.

Providing a single source of truth regarding the amount of virtual currency owned by an entity is challenging in a decentralized setting. Suppose the problem of validating credentials in a decentralized virtual currency system has somehow been solved. In order to reliably to store the amounts of virtual currency owned by an entity in the network, these amounts would have to stored in multiple network nodes. This is to avoid single points of failure. Due to packet propagation delays in the peer-to-peer network, there will be some delay before all the nodes in the network see the update to an amount. This delay can be exploited by a malicious entity to perform a double spend.

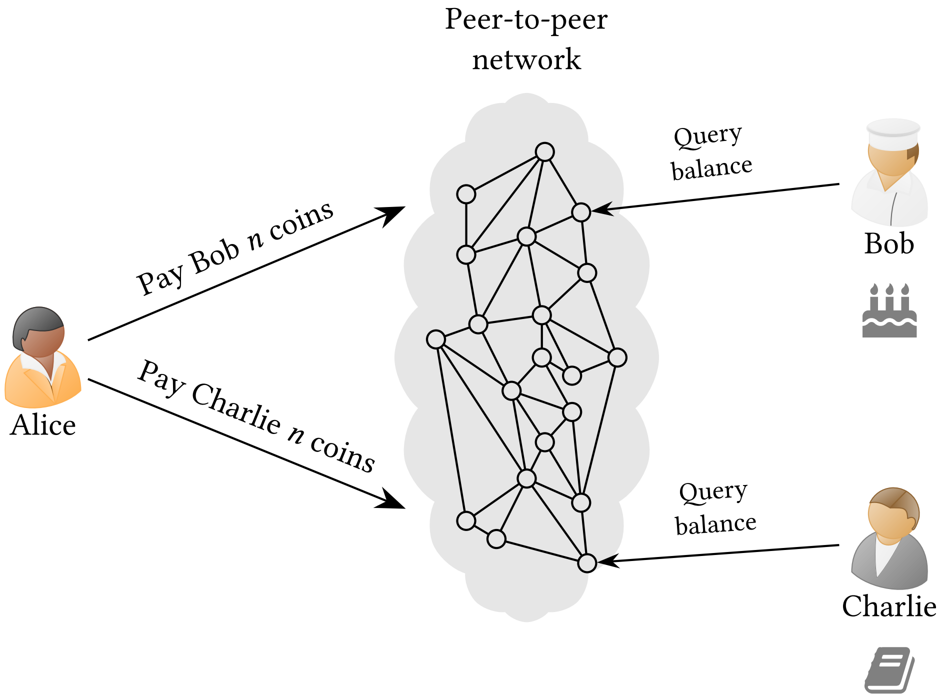


Figure 1.1: Illustration of the double spending problem

Consider the situation in Figure 1.1 where Alice wants to use a decentralized virtual currency to purchase a cake from Bob and a book from Charlie. For simplicity, let us assume that these items both cost n digital coins. Bob and Charlie will provide Alice with their wallet addresses where they expect to receive the coins. The balance in a wallet address can be obtained by querying the peer-to-peer network. Alice creates two transactions which *both spend the same n coins* which Alice owns — one paying Bob for the cake and the other paying Charlie for the book. These two transactions conflict and a network node will only accept one of them. But the network consists of several nodes, some of which can be made to accept one of the transactions and the rest can be made to accept the other. So Alice can transmit the transaction paying Bob to the portion of the network Bob is connected to and the transaction paying Charlie to the portion of the network Charlie is connected to. When Bob and Charlie query the balances in their wallet addresses immediately after Alice's transmissions they will see a valid payment. They will eventually become aware of the conflicting transactions. But if they part with their goods before the conflicting transactions reach them, Alice would have successfully executed a double spend.

While the above attack can potentially be mitigated by having Bob and Charlie make connections to several nodes in the network, Alice can mount a similar attack by taking control of the subset of nodes either Bob or Charlie is connected to. The main takeaway is that the double spending problem needs to be addressed for the correct operation of a decentralized virtual currency.

1.1.3 Controlling Currency Supply

The supply of fiat currency notes and coins is determined by the respective governments. They decide the quantity and schedule of new currency creation. In a decentralized currency system, trusting any one entity to determine the currency supply is not feasible. Such an entity may disappear or abuse their elevated status for their own benefit. Even if all the network nodes agree on a policy for currency creation, preventing the counterfeiting of virtual currency is challenging as digital data can be perfectly copied.

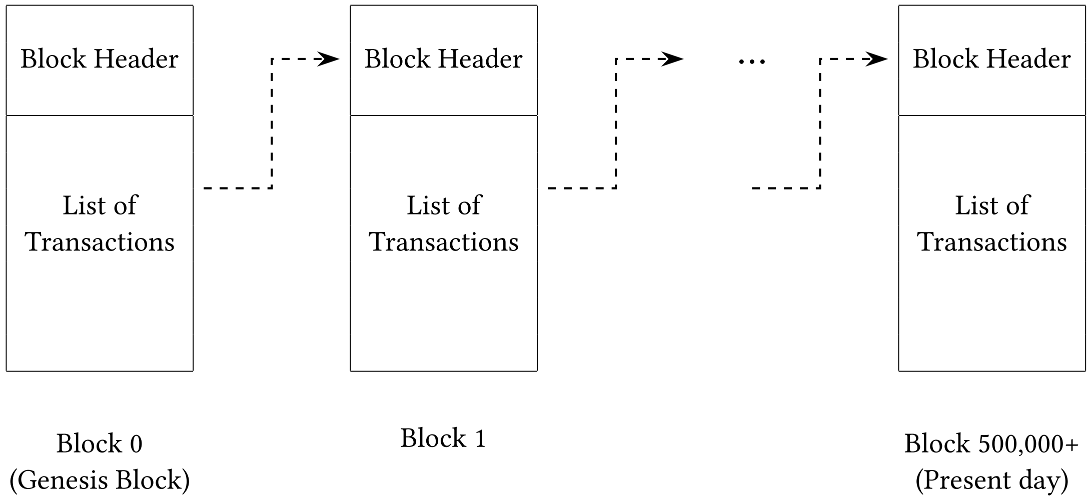


Figure 1.2: Illustration of the blockchain

1.2 The Design of Bitcoin

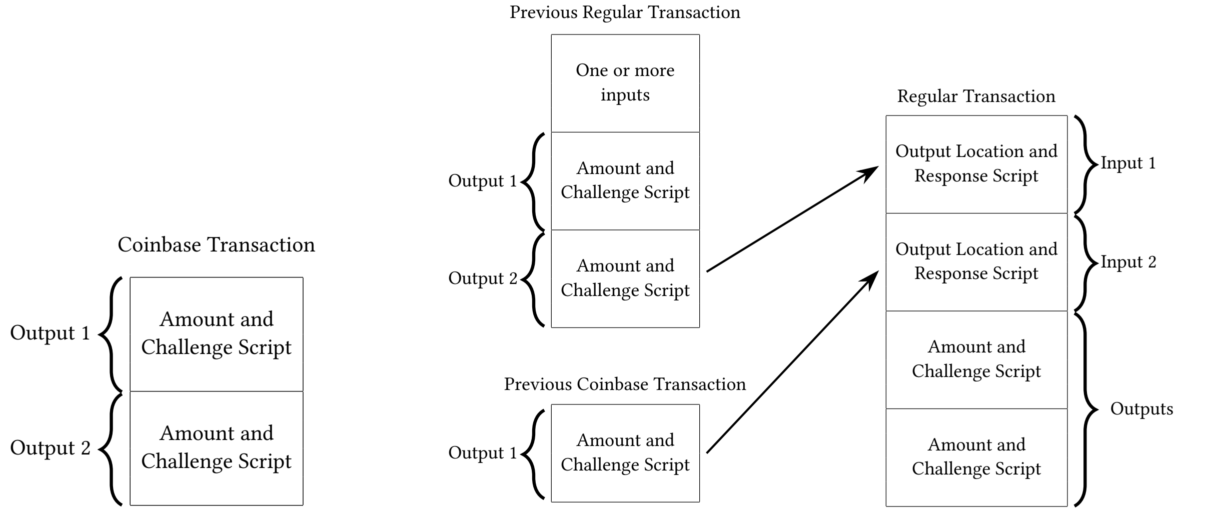
Bitcoin solves the challenges described in the previous section using a combination of cryptography and incentive engineering. We give a brief account of the design of the Bitcoin protocol before discussing its security.

1.2.1 Transactions and the Blockchain

The fundamental unit of state in the Bitcoin protocol is a *transaction*. At a high level, a Bitcoin transaction encodes the transfer of a certain amount of bitcoins from a sender to a receiver. All the transactions which have occurred since the beginning of the Bitcoin protocol operation are stored in a public database called the *blockchain*. It is a linear list (or chain) of blocks, where a block consists of a header followed by a list of transactions (as illustrated in Figure 1.2). The first block which was created in January 2009 is called the *genesis block*. Currently, the blockchain has more than half a million blocks with an average of six new blocks being added to it every hour. Network nodes called *full nodes* maintain a copy of the blockchain on their local hard disks. When a new block is added to the blockchain, it is broadcast to all nodes in the network enabling the full nodes to update their local copies of the blockchain. The logical connection (represented by dashed lines in Figure 1.2) between consecutive blocks in the blockchain is created by inserting the *cryptographic hash* of a block's contents into the block header of the next block.[4]

Each block can have a different number of transactions but must necessarily contain a *single* special transaction called the *coinbase transaction*, where new bitcoins are created. The amount of new bitcoins created is called the *block subsidy* and it is currently 12.5 bitcoins per block. These newly created bitcoins are stored in the *outputs* of the coinbase transaction. Each output consists of an amount of bitcoins and a *cryptographic challenge* (as illustrated in Figure 1.3a). In general, the cryptographic challenges are described using a simple programming language called *Script* which was developed by Satoshi Nakamoto specifically for Bitcoin. Consequently, the cryptographic challenge is called a *challenge script*. An example of a cryptographic challenge is a request for a digital signature which can be validated using a public key. Only the holder of the corresponding private key can correctly answer this challenge (by generating the digital signature). The bitcoins stored

[4] More details about the cryptographic hash and the reason behind embedding it in the block header will be given in Section 1.2.3 on Bitcoin mining and Section 2.3.2 on confirmed transaction tampering.


Coinbase Transaction
Output 1
Amount and Challenge Script
Output 2
Amount and Challenge Script
Previous Regular Transaction
One or more inputs
Output 1
Amount and Challenge Script
Output 2
Amount and Challenge Script
Previous Coinbase Transaction
Output 1
Amount and Challenge Script
Regular Transaction
Output Location and Response Script
Input 1
Output Location and Response Script
Input 2
Amount and Challenge Script
Amount and Challenge Script
Outputs

(a) A coinbase transaction with two outputs

(b) A regular transaction (shown on the right) with two inputs and two outputs

Figure 1.3: Structure of coinbase and regular transactions

in an output can be spent by anyone who can correctly answer the cryptographic challenge in the output.

The coinbase transaction in a block is followed by zero or more *regular transactions*.[5] Regular transactions have both *inputs* and *outputs*. Figure 1.3b shows a regular transaction (on the right side) which has two inputs and two outputs. The format of the outputs is the same as before — each output contains an amount and challenge script. Each input contains the location of a previous output in the blockchain and a *response script* which correctly answers the cryptographic challenge encoded in that output's challenge script. By answering the cryptographic challenge in a previous output, an input *unlocks* the bitcoins stored there. In Figure 1.3b, the first input of the regular transaction unlocks the bitcoins in the second output of a previous regular transaction and the second input unlocks the bitcoins in the first output of a previous coinbase transaction. So the inputs of a regular transaction represent the *sources* of bitcoins which are being transferred. The *destinations* of these bitcoins are represented by the outputs in the regular transaction. The bitcoins in these outputs can be later spent by anyone who can answer the corresponding cryptographic challenges.

The sum of the amounts locked in the outputs of a regular transaction is slightly less than the sum of the amounts unlocked by the inputs. The difference is called the *transaction fees*. It is collected by special network nodes called *miners* who perform the task of adding new regular transactions to the blockchain.

Note that the regular transaction inputs do not contain a withdrawal amount. This is because there is no way to partially spend the amount of bitcoins in an output. By providing a correct response to an output's challenge script, *all the bitcoins* in the output are unlocked for spending. Consequently, a transaction output can only be in two states — *unspent* and *spent*.

While an output cannot be partially spent, it is still possible to transfer only a portion of the output's bitcoins to a receiver. For example, suppose Alice can unlock a previous transaction output containing 3 bitcoins. She wants to send only 1 bitcoin to Bob. She creates a regular transaction which has one input and two outputs. The input unlocks Alice's previous output containing the 3 bitcoins. The first output in the new regular transaction has an amount field equal to 1 bitcoin

[5] If the block contains no regular transactions, it is called an *empty block*. While empty blocks are currently rare, they were quite common in the early days of Bitcoin (when it was a relatively unknown system).

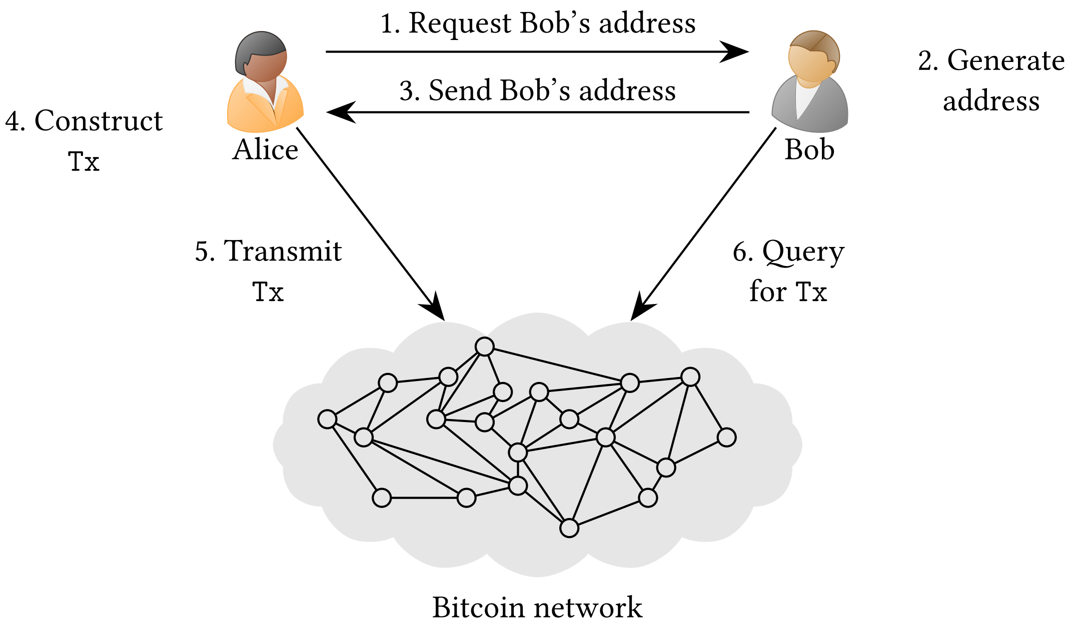


Figure 1.4: Transaction workflow in Bitcoin

and a challenge script which can only be unlocked by Bob. Alice obtains such a challenge script by requesting Bob to send it before creating the transaction. The second output has an amount field equal to 1.9999 bitcoins (assuming a transaction fee of 0.0001 bitcoins) and a challenge script which can only be unlocked by Alice. As the second output represents a transfer back to Alice, it is called a *change output*. When the new regular transaction is added to the blockchain, Bob would have gained 1 bitcoin and Alice would retain control of 1.9999 bitcoins out of the initial 3 bitcoins (albeit in a different transaction output).

1.2.2 Bitcoin Ownership

There is no concept of accounts in the Bitcoin system. Ownership is determined by the ability to correctly answer the cryptographic challenges stored in *unspent transaction outputs (UTXOs)*.[6] The total amount of bitcoins owned by a user is the sum of the amounts in all the UTXOs the user can unlock.

Bitcoin relies on *asymmetric cryptography* to validate ownership in a decentralized manner. Each user creates a pair of keys, a public key and a private key. Challenge scripts can be generated using only the public key but generation of the corresponding response script needs the private key. For example, suppose Alice wants to transfer some bitcoins to Bob in return for some goods or services. The transaction workflow has the following steps which are illustrated in Figure 1.4.

1. Alice asks Bob to send her his *Bitcoin address* which is an alphanumeric string that can be transformed into a challenge script.

2. Bob generates a public-private key pair and generates a Bitcoin address from the public key.

3. Bob sends the generated address to Alice.

4. Alice creates a transaction Tx which contains one input and one output. The input contains the location (in the blockchain) of a UTXO which Alice can unlock and the response script

[6] It is common practice in the Bitcoin community to abbreviate the word *transaction* as TX or Tx.

which correctly answers the challenge script in that UTXO. The output contains the challenge script shared by Bob and an amount of bitcoins which Alice wishes to transfer to Bob.

5. Alice then transmits `Tx` onto the Bitcoin peer-to-peer network. Nodes which receive `Tx` will verify that the response script Alice used to unlock the UTXO is valid. If it is invalid, the network nodes will reject `Tx`. If the response script is valid, they will relay `Tx` to their neighboring nodes. `Tx` will eventually reach miner nodes in the network who will add it to a new block on the blockchain.

6. Bob will query the blockchain to check if a payment to his address has appeared on it. Bob can do this either by running a Bitcoin node himself or by connecting to a node in the network. Once Bob sees `Tx` in a new block on the blockchain, he will wait for the blockchain to grow past the block containing `Tx` by a few blocks before he provides goods or services to Alice. Bob does this to avoid becoming the victim of a double spending attack by Alice (this is explained further in Section 1.2.4).

In the second step described above, Bob need not necessarily generate a new address for each transaction. He could reuse an address which he had generated for a previous transaction but such *address reuse* is not recommended. If a unique Bitcoin address is used to receive bitcoins in each transaction, a different private key will be required to spend the different UTXOs. So the damage caused by the loss or theft of a single private key can be limited to the loss of bitcoins stored in a single UTXO. But if the same address is used to receive bitcoins in multiple transactions, the loss of the private key results in the loss of bitcoins stored in all the corresponding UTXOs.

1.2.3 Bitcoin Mining

The Bitcoin blockchain is a public database of transactions. To be successful as an open payment system, anyone should be allowed to add transactions to the blockchain. But at the same time, addition of fraudulent transactions to the blockchain needs to be prevented. An example of a fraudulent transaction is one which attempts to double spend some bitcoins. Even if methods to detect fraudulent transactions are available, there is no guarantee that people will spend computing and network resources to run network nodes which faithfully execute these methods instead of colluding with the fraudster. So there is a need to incentivize people to check for and reject fraudulent transactions.

Bitcoin mining is the procedure by which new blocks are added to the blockchain while ensuring that these blocks contain only valid transactions. Additionally, the mining procedure controls the rate of creation of new bitcoins. Before we can describe the mining workflow, we need to understand some properties of *cryptographic hash functions*.

1.2.3.1 Cryptographic Hash Functions

Cryptographic hash functions take as input a bitstring of arbitrary length and output a bitstring of fixed length. For example, the SHA-256 hash function which is used in Bitcoin can take as input a bitstring of length less than 2^{64} bits and outputs a bitstring of length 256 bits. While a formal definition of cryptographic hash functions is outside the scope of our current discussion, these functions satisfy two properties:

1. **Difficult to invert**: Given the output of a cryptographic hash function, it is difficult to recover the input which led to this output. Suppose H is a cryptographic hash function. For an input bitstring x, let $y = H(x)$ be the output bitstring. It is difficult to recover x given only y even if the algorithm used to compute y from x is known. The difficulty of recovery is from a

Input	SHA-256 Output
bitcoin0	2277efd2e9051a1978682cad7a111876031f7fcdb9a2a06b5fdeee160dd8f34e
bitcoin1	dbdbac0b3072d7677fc94eebaf8eba9e81e5c3b7de6899dae12c98d6799b065a
bitcoin2	1ed7259a5243a1e9e33e45d8d2510bc0470032df964956e18b9f56fa65c96e89
bitcoin3	0c5582329503f93b4b243a986551d9e22e46ee9ba681d687078cbcbad0c7d023
bitcoin4	0a49508bf91ac4f98e6a01b575e1a3f200a5d9a03d00219aea52b15b064cdf50
bitcoin5	de6206bd52f4228ebc556c85b26e3582fa141f8839a11d2a2ca761d0f7e24ec3
bitcoin6	e1abb7b46d14bb2c3e13208ebc9790ab847f6b5265adbf154d4200b513359e22
bitcoin7	c07bed0fae2067f2ed35cc443d97aeacbaf0b59dcbd619f76c75477690b82d3b
bitcoin8	8ecc8a5ebc2a99db8e950c29242e7052ae2930cd60258176efe36750a4e33170
bitcoin9	38ab2bcafbf65eb6204162d28082ad7616f2a66f20b27696262e3842b3712d0b

Table 1.1: Illustrating the pseudorandom behaviour of the SHA-256 hash function

computational perspective. It *is possible* to recover x from y by repeatedly computing $H(z)$ for different values of z until the output matches y. But the amount of computation involved is prohibitively large due to the large number of possibilities for z. For example, if we restrict our search to only input bitstrings z of length 100 bits then there are 2^{100} possibilities for z. If computing a single value of $H(z)$ takes a nanosecond, searching through all 2^{100} values will take 78 billion years. The fact that the input can be several hundred bits long makes the problem even more difficult.

2. **Pseudorandom outputs**: For two inputs to cryptographic hash function which differ even in a single bit, the respective outputs are very different. While the cryptographic hash functions are deterministic functions (always returning the same output for the same input), the outputs for different inputs appear as if they were randomly chosen. This is illustrated in Table 1.1 where the second column shows the output of the SHA-256 hash function for the input in the first column.[7] Even though the input strings differ only in the last character, the outputs bear no resemblance to each other.

Cryptographic hash functions are constructed by specifying a complicated input-output relationship where each bit of the input affects all the output bits. Even a single input bit change results in a wildly different output. While it is easy to go in the forward direction from input to output, it is difficult to go in the reverse direction from an arbitrary output to an input.

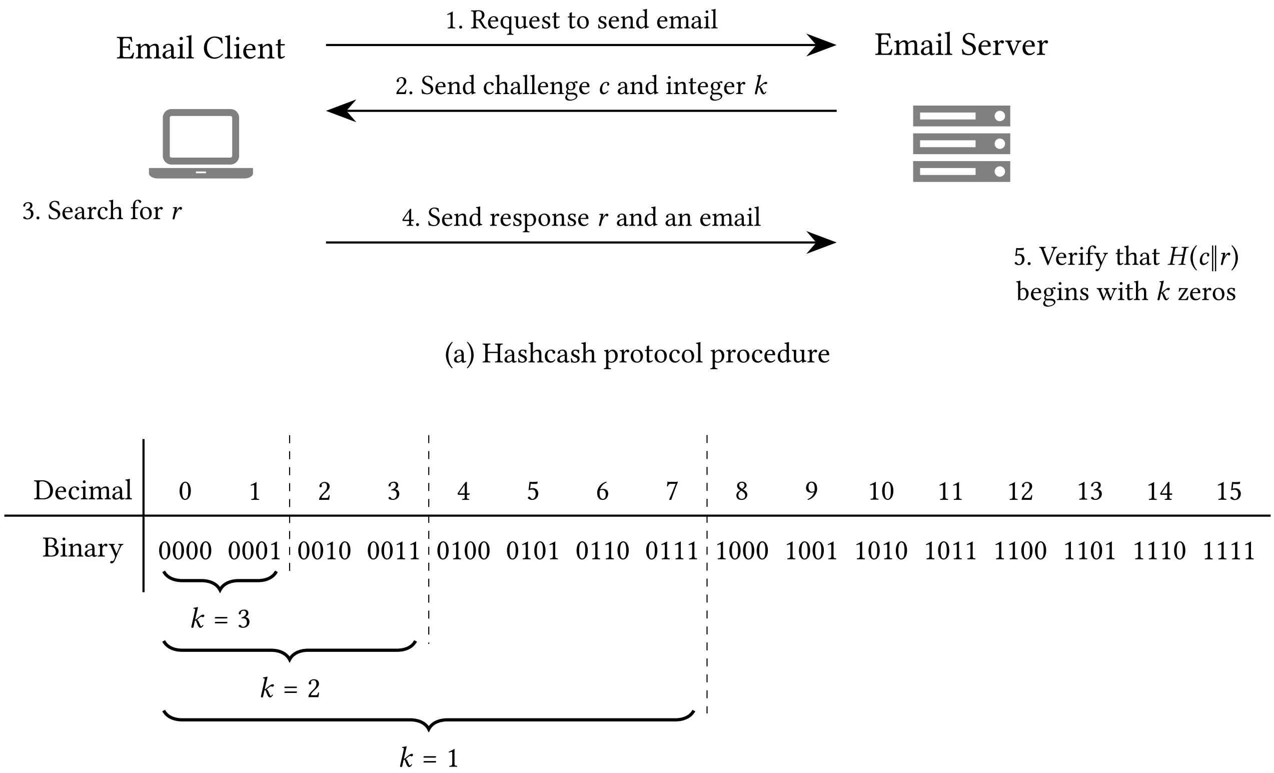
1.2.3.2 Hashcash

Hashcash is a scheme proposed in 1997 to combat email spam. It is relevant to Bitcoin because the *proof-of-work (PoW)* idea from Hashcash is used in Bitcoin mining. Suppose an email server receives emails from several email clients and it wants to reduce the number of spam emails it receives from the clients. The server advertises a specific cryptographic hash function H that it will use in the the Hashcash protocol. Let n be the length (in bits) of the outputs of the hash function H. The protocol has the following steps which are illustrated in Figure 1.5a.

1. A client connects to the server and requests to send a email to it.

2. The server sends the client a challenge string c and an integer k less than n.

[7] The outputs are represented using in hexadecimal digits (base 16). Each hexadecimal digit represents 4 bits and there are 64 hexadecimal digits for each of the 256-bit outputs.


Email Client
1. Request to send email
Email Server
2. Send challenge c and integer k
3. Search for r
4. Send response r and an email
5. Verify that $H(c\|r)$ begins with k zeros

(a) Hashcash protocol procedure

Decimal	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Binary	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111

(b) Hashcash-related subsets of n-bit strings for $n = \mathbf{4}$ and $k = \mathbf{1}, \mathbf{2}, \mathbf{3}$

Figure 1.5: Illustration of Hashcash protocol and proof-of-work

3. Client searches for a response string r such that $H(c\|r)$ begins with k zeros, where $c\|r$ denotes the concatenation of the strings c and r.

4. Once such an r is found, the client sends it to the server along with an email.

5. The server verifies that $H(c\|r)$ begins with k zeros. If it does, the server accepts email from the client. Otherwise, the email is rejected.

The rationale behind this protocol design is as follows. Email spammers send millions of emails each day. To ensure that these emails are accepted by a server implementing the Hashcash protocol, a spammer's client would have to find an r such that $H(c\|r)$ begins with k zeros. The server prevents the client from reusing r by changing the challenge c for every email. Since H is a cryptographic hash function, it is difficult to invert and the client has no option but to compute $H(c\|r)$ for different values of r until it finds an output beginning with k zeros. As this work has to be performed for every email, the rate at which a spammer can send emails is reduced.

If the output length of H is n bits, the first k bits are constrained to be zero and the remaining $n - k$ bits can each be either 0 or 1. So the client needs to find a value r such that $H(c\|r)$ is equal to one of 2^{n-k} values. For the sake of illustration, consider the case when the hash function output length is $n = 4$ bits. In Figure 1.5b, for $k = 3$ the curly braces mark the two 4-bit strings 0000 and 0001 which begin with at least three zeros. Similarly, for $k = 2$ and $k = 1$ the corresponding curly braces mark the 4-bit strings which begin with at least two zeros and one zero respectively. As H has pseudorandom outputs, the probability of success in a single trial is

$$\frac{2^{n-k}}{2^n} = \frac{1}{2^k},$$

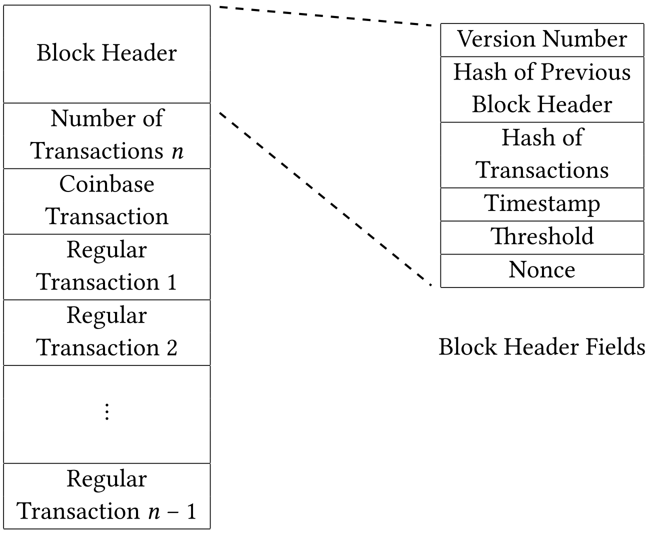


Figure 1.6: Block and block header formats

which decreases exponentially with k. This is similar to the problem of repeatedly tossing a coin n times until the first k tosses out of the n tosses all result in a heads. On the average, the client has to try 2^k different values of r until it succeeds.

The response r corresponding to a challenge c is considered a proof-of-work (PoW). There is *asymmetry* between PoW generation and verification. PoW is difficult to generate for large k as it requires 2^k hash function evaluations on the average. But it is easy to verify as the server needs to perform only a single hash function evaluation using the value r sent by the client.

Note that condition of a bit string beginning with at least k zeros can be expressed as the magnitude of the corresponding decimal number falling below a *threshold*. For example, if the decimal representation of a 4-bit string is less than 2 then it begins with at least three zeros (as illustrated in Figure 1.5b). If it is less than 4, then it begins with at least two zeros; if it is less than 8 it begins with at least one zero. This threshold interpretation will be useful in understanding Bitcoin mining.

1.2.3.3 Mining Workflow

As mentioned before, Bitcoin mining is the process by which new blocks are added to the blockchain and as a byproduct new bitcoins are created. The mining workflow is as follows.

1. People who wish to perform Bitcoin transfers will transmit new transactions onto the network. Each transaction includes a small transaction fee which is equal to the difference between the number of bitcoins unlocked by the transaction inputs and the number of bitcoins being stored in the transaction outputs.

2. Special nodes called miners collect some of these transactions into a *candidate block*. Each miner will construct its own candidate block.

   (a) The candidate block consists of a block header followed by a list of transactions as illustrated on the left of Figure 1.6.

   (b) The first transaction in the block is a coinbase transaction. Each miner will specify itself as the receiver of the bitcoins in the coinbase transaction. It does this by using its own Bitcoin address to generate the challenge scripts in the coinbase transaction outputs.

   (c) While the destinations of the bitcoins being transferred are explicitly specified in the outputs of the coinbase transaction, the source of bitcoins is implicit. The number of bitcoins being transferred to the outputs in a coinbase transaction is called the *block reward*. It is equal to the sum of the transaction fees paid by all the regular transactions in the block plus the block subsidy. The block subsidy is currently equal to 12.5 bitcoins per block and represents newly created bitcoins. These new bitcoins will come into existence under the ownership of the miner if the candidate block is successfully added to the blockchain.

(d) The size of a block is limited to about 4 megabytes. Each regular transaction occupies a few hundred bytes. As the miner stands to gain the transactions fees from all the regular transactions in the block, it will prioritize transactions which pay the highest fee per byte while constructing its candidate block.

(e) The block header consists of the following six fields:

   i. A block version number.

   ii. The double SHA-256 hash of the block header of the latest block in the blockchain.[8]

   iii. The double SHA-256 hash of all the transactions in the block (including the coinbase transaction).

   iv. A timestamp.

   v. A threshold.

   vi. A *nonce*.[9]

3. While the first five fields in the block header are determined by the state of the blockchain, the transactions included in the block, and the protocol rules, the miner is free to set the nonce to any value. Each miner will try to find a nonce such that the double SHA-256 hash of the its candidate block header falls below the threshold, i.e.

$$\text{SHA256}(\text{SHA256}(\underbrace{\text{Version Number} \parallel \cdots \parallel \text{Nonce}}_{\text{Candidate Block Header}})) \leq \text{Threshold}.$$

The miner's task is similar to the email client's task in the Hashcash scenario. Here the first five fields of the block header form the challenge c and the nonce found by the client corresponds to response r. In the Hashcash scenario, the server generated the challenge c while in Bitcoin mining the challenge is generated from the blockchain and network state.

4. Multiple miner nodes will compete to find the PoW nonce. The miner node which succeeds in finding such a nonce is said to have *mined a new block*. The successful miner node immediately broadcasts the new block on the network.

5. When a new block is received by a network node, if the transactions in it pass validity checks then it is appended to the node's local copy of the blockchain. In addition to doing this, miner nodes abandon their current candidate blocks and begin mining on top of the new block. This is illustrated in Figure 1.7 which shows the state of the mining process before and after a new block is mined. Initially, the latest block in the blockchain is block B as shown in the left side of the figure. Suppose there are only miner nodes in the network, each one trying to mine candidate blocks C_1, C_2, and C_3 respectively. Suppose the miner mining C_2 is successful in finding a nonce such that the hash of the block header falls below the threshold. The miner adds C_2 to its copy of the blockchain and begins mining candidate block D_2. It also broadcasts C_2 on the network which causes the other two miners to abandon their candidate blocks and begin mining candidate blocks D_1 and D_3.

[8] The double SHA-256 hash of some input x is obtained by calculating SHA256(SHA256(x)).

[9] In cryptography, the word nonce is used to denote a number which is used only once.

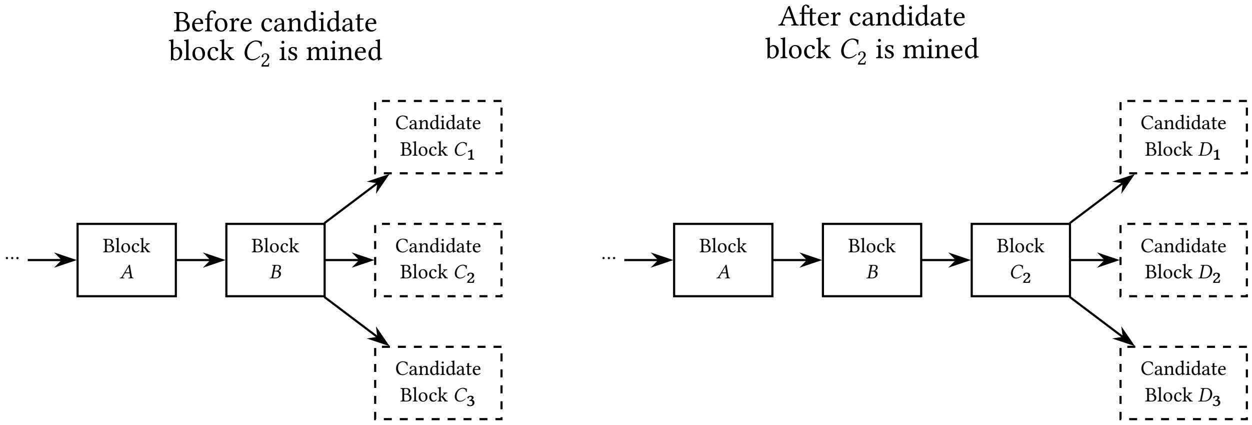


Figure 1.7: Illustration of miner behaviour before and after a new block is mined

The main innovation in the Bitcoin protocol is the incentivization of new block addition via the block reward. The desire to earn the block reward encourages the following miner behaviours.[10]

- Miners will try to pack as many transactions as possible in a candidate block in order to maximize the transaction fees they will earn. This benefits users who see their transactions being included in blocks faster. Including more transactions in a block does not change the difficulty of the PoW problem as it is independent of the size of the block. The miner only hashes the block header which always has a fixed size of 80 bytes.

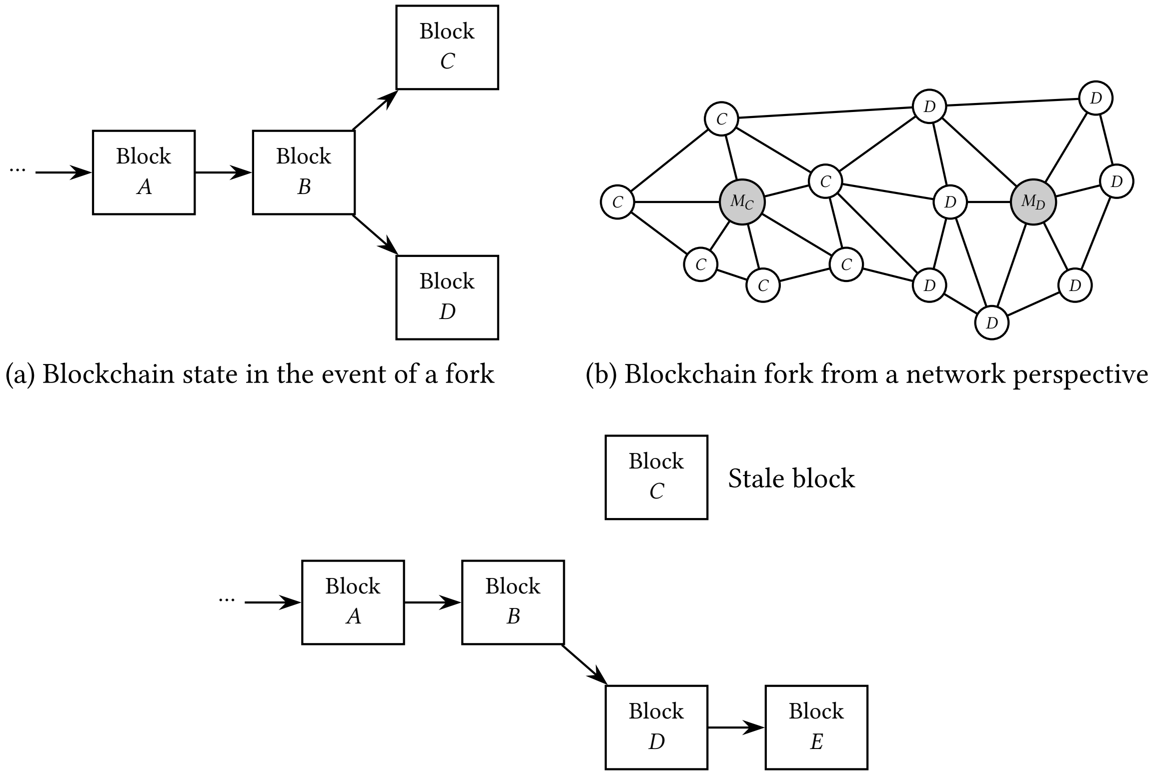
- The coinbase transaction containing the block reward is considered valid only after it appears in a block on the blockchain. This ensures that miners do not knowingly include invalid transactions in their candidate blocks. An example of an invalid transaction is one which tries to use an *already spent* transaction output as a source of bitcoins (i.e. a double spending transaction). If a miner were to include an invalid transaction in its candidate block, the network nodes will reject this block even if it satisfies the PoW condition and the coinbase transaction containing the miner's block reward will not be added to the blockchain. A scenario where a miner unknowingly includes a double spend transaction in its candidate block is described in Section 1.2.4 along with a description of how such a transaction is eventually removed.

- When miners receive a new block which indicates that they have lost the race to find the next block on the blockchain, they immediately abandon their current candidate blocks and begin mining on top of the new block. If they were to ignore the new block and continue mining their candidate blocks, they will be missing the opportunity to win the next block reward. This reasoning assumes that a single miner does not control 50% or more of the *total network hash rate*, which is defined as the number of block header hashes all the miners in the network can calculate per second. In next chapter, we discuss the case when a miner does control the majority of the network hash rate.

1.2.3.4 Achieving Consensus

It is possible that two miners mine their respective candidate blocks at almost the same time. Each of the miners may broadcast their block onto the network before becoming aware of the other miner's block. This causes a *fork* in the chain as there are two possible successors to the latest block. This is

[10] These behaviours are encouraged in the sense that deviating from them will reduce the block rewards which the miner can earn.



(a) Blockchain state in the event of a fork

(b) Blockchain fork from a network perspective

(c) Blockchain state when the lower branch accumulates more work than the upper branch

Figure 1.8: Achieving consensus after a blockchain fork

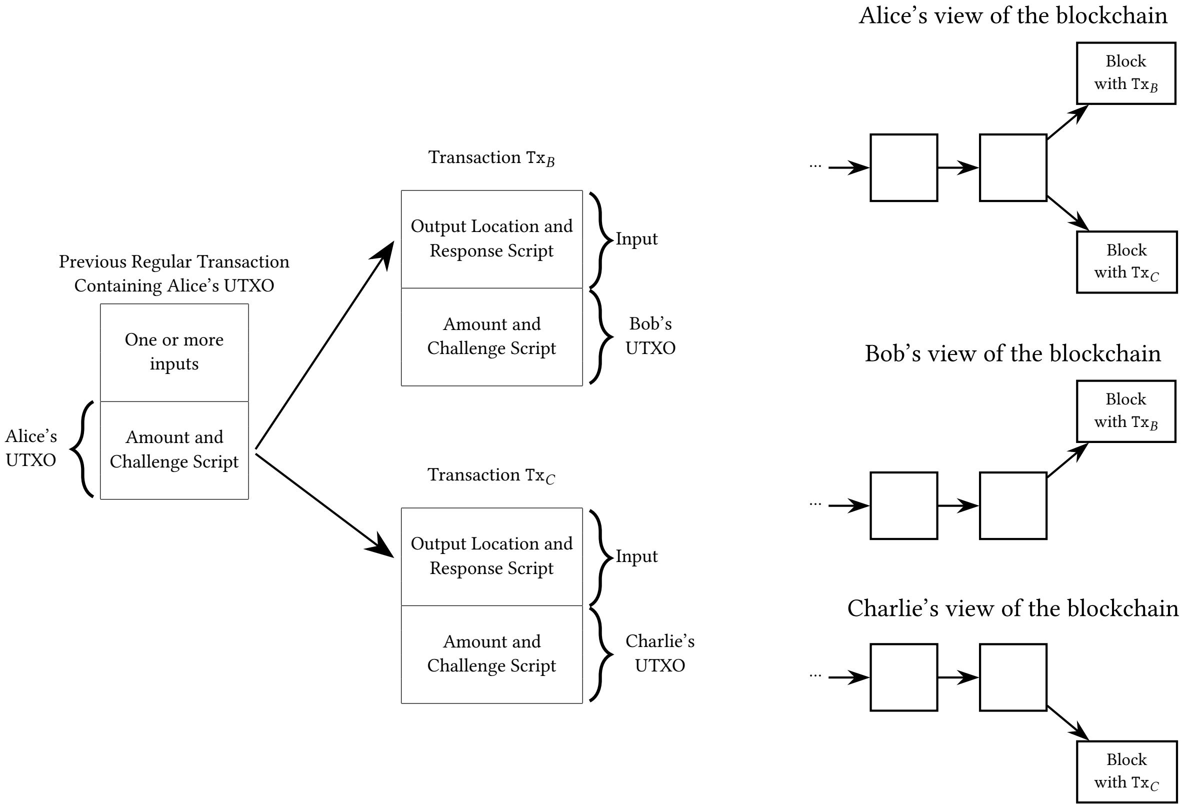
illustrated in Figure 1.8a where blocks C and D are both successors of block B. When presented with the blocks C and D, network nodes will accept the block that arrived first and reject the second one. Figure 1.8b illustrates the network perspective when miner nodes M_C and M_D, which mined blocks C and D respectively, broadcast their blocks at the same time. The nodes neighboring M_C accept C and nodes neighboring M_D accept D. This is shown by labeling the nodes by the block they consider to be the successor of block B (which is either C or D). The miners which consider block C to be the right successor will begin mining on top of it. Similarly, some miners will begin mining on top of block D.

Due to the random nature of the mining process, one of the chains in the fork will accumulate more work (by having more blocks). The Bitcoin protocol mandates that nodes should always *switch to the chain which required the most work to produce.* Figure 1.8c illustrates the case when the miners mining on top of block D mined a new block E before the miners mining on top of block D. When the block E is broadcast on the network, the miner nodes mining on top of block C will abandon the upper chain and switch to mining on top of block E. The abandoned block C becomes a *stale block.* The coinbase reward in block C is invalid as it is not part of the blockchain. Any regular transactions present in block C which were not included in blocks D and E will be retried.

This mechanism of resolving forks is called *proof-of-work consensus.* Forks will be resolved in favor of the chain which has the majority of the network hash rate working on it. If nodes misbehave by not switching to the chain with the most work, they stand to lose future block rewards by mining blocks which are considered stale by the rest of the network.

1.2.4 Preventing Double Spends During Forks

In the absence of chain forks, double spending is prevented by the miners when assembling candidate blocks. They do not include transactions which try to spend an already spent transaction output


Previous Regular Transaction
Containing Alice's UTXO
One or more inputs
Alice's UTXO
Amount and Challenge Script
Transaction Tx$_B$
Output Location and Response Script
Input
Amount and Challenge Script
Bob's UTXO
Transaction Tx$_C$
Output Location and Response Script
Input
Amount and Challenge Script
Charlie's UTXO
Alice's view of the blockchain
Block with Tx$_B$
Block with Tx$_C$
Bob's view of the blockchain
Block with Tx$_B$
Charlie's view of the blockchain
Block with Tx$_C$
...

(a) The two transactions created by Alice

(b) The different views of the blockchain state

Figure 1.9: Double spending attack during a blockchain fork

in their candidate blocks. But in the event of a fork, the state of the blockchain is temporarily inconsistent across the network. An attacker can exploit this fleeting inconsistency to execute a double spend.

To illustrate the attack, let us revisit the double spending attack described in Section 1.1.2 and illustrated in Figure 1.1. Alice wants to purchase a cake from Bob and a book from Charlie. Both Bob and Charlie are willing to accept payment in bitcoins. Alice creates two transactions Tx$_B$ and Tx$_C$ both having a single input and a single output. The input in both the transactions unlocks a UTXO owned by Alice in a previous regular transaction (as illustrated in Figure 1.9a). The output in Tx$_B$ creates a UTXO which only Bob can unlock and the output in Tx$_C$ creates a UTXO which only Charlie can unlock. The transactions Tx$_B$ and Tx$_C$ are conflicting and cannot simultaneously exist in a single chain. But suppose Alice can induce a fork in the blockchain satisfying two conditions:

1. One chain in the fork has a block containing Tx$_B$ and the other chain has a block containing Tx$_C$ as shown in the blockchain state on the top of Figure 1.9b.

2. Bob receives the block containing Tx$_B$ first and considers the upper chain to be the valid chain. On the other hand, Charlie receives the block containing Tx$_C$ first and considers the lower chain to be the valid chain. This is illustrated in the middle and bottom blockchain states shown in Figure 1.9b.

Alice could induce such a fork by broadcasting the transactions Tx$_B$ and Tx$_C$ to the miners located near Bob and Charlie respectively and the hope that they appear in two different chains of a fork. Or Alice may herself be a miner who mines two different candidate blocks containing Tx$_B$ and Tx$_C$, and proceeds to broadcast these blocks to Bob and Charlie respectively.

With the unresolved blockchain fork in place, Alice can request Bob and Charlie to part with their goods. When Bob queries the blockchain to check for Alice's payment, he finds Tx$_B$ present. And when Charlie does the same, he finds $\mathtt{Tx}_C$ present in the blockchain. If Bob and Charlie were to supply the cake and book immediately, Alice would have successfully executed a double spend. When the blockchain fork is eventually resolved, either the chain containing $\mathtt{Tx}_B$ or the chain containing $\mathtt{Tx}_C$ will survive and the other chain will become invalid. If the chain containing $\mathtt{Tx}_B$ survives, Bob would have received the payment from Alice but Charlie would have been cheated. While Alice's UTXO is spent, she has gained two items for the price of one.

To avoid such a double spending attack, merchants typically wait for a few *transaction confirmations* before providing goods or services in return for bitcoins. When a transaction Tx appears in a block on the blockchain, it is said to have received *one confirmation.* If a new block is added to the blockchain which succeeds the block containing Tx, it is said to have received *two confirmations.* Merchants typically wait for six confirmations before considering a payment in Tx to be valid. The rationale is that after six confirmations the probability of the transaction becoming invalid via a fork resolution is small. The number six is a heuristic and merchants can choose to wait for more or less than six confirmations depending on the value of the goods or services they are providing.

1.2.5 Bitcoin Supply

The supply of new bitcoins is controlled by regulating the rate at which new blocks are mined. Recall that new bitcoins are created in the coinbase transaction of a new block which is added to the blockchain. The difficulty of finding a new block depends on the threshold used in the mining process. If the threshold is increased, mining a new block becomes easier as the number of allowed double SHA-256 output values increases. Decreasing the threshold has the opposite effect, i.e. mining a new block becomes harder. The Bitcoin protocol aims to have an *average inter-block time of 10 minutes.* The mining threshold is adaptively adjusted every 2,016 blocks to achieve this 10 minute inter-block time. The number 2,016 was chosen because it represents the number of blocks that will be mined in two weeks if the inter-block time is exactly 10 minutes (2,016 blocks = 14 days × 24 hours/day × 6 blocks/hour).

To change the threshold, the timestamps in the block headers are used to calculate the amount of time taken by the network to mine 2,016 blocks. The new threshold is obtained from the old threshold as follows.

$$\text{New Threshold} = \frac{\text{Number of seconds taken to mine 2,016 blocks}}{\text{Number of seconds in two weeks}} \times \text{Old Threshold.}$$

If the 2,016 blocks were mined in less than two weeks, then the new threshold is smaller than the old threshold. This makes the mining problem harder for the whole network and increases the time between new blocks. If mining 2,016 blocks took more than two weeks, then the new threshold is larger than the old threshold. This makes the mining problem easier and decreases the time between new blocks. Such an adaptive procedure is required because the total amount of network hash rate mining new blocks varies over time. It increases when new miners begin mining operations or when old miners add more resources for mining. It decreases when miners cease or reduce their mining operations.

The block subsidy was set to 50 bitcoins at the start of the Bitcoin network operation in 2009. The Bitcoin protocol mandates that the block subsidy *be halved after every 210,000 blocks*, which is approximately every 4 years. The block subsidy reduced to 25 bitcoins in November 2012 when block 210,000 was mined and to 12.5 bitcoins in July 2016 when block 420,000 was mined. The smallest indivisible unit of the Bitcoin currency is called a *satoshi* with each bitcoin being equal to *100 million satoshi.* As the block subsidy is progressively halved, it will eventually become less than 1 satoshi. At this point, it will be considered zero. The block subsidy will become zero when block 6,930,000 is

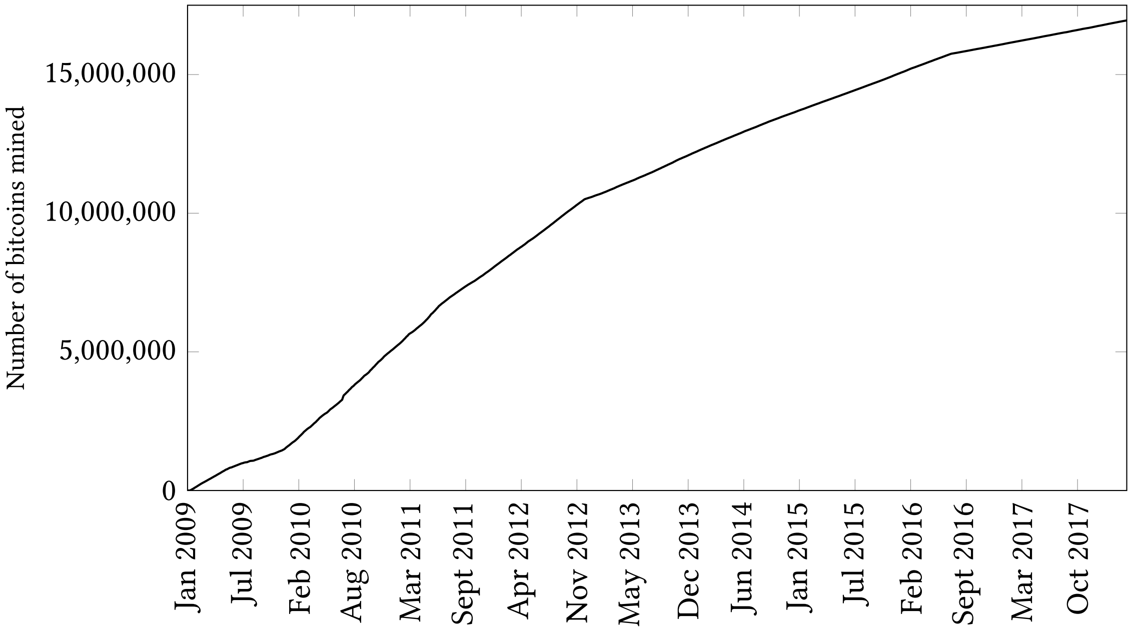


Figure 1.10: Cumulative number of bitcoins mined over time.

mined which is expected to be around the year 2140. As the rate of new bitcoin creation decreases geometrically, the total number of bitcoins which will ever come into existence is about *21 million*. Figure 1.10 shows the cumulative number of bitcoins mined over the years since 2009.[11] As of now, about 17 million bitcoins have already been mined. Once all the bitcoins have been mined, miners will be incentivized to mine new blocks solely by the transaction fees offered by regular transactions in the block.

1.3 Summary

The Bitcoin blockchain contains a record of all past and current ownership of all the bitcoins which have come into existence. Ownership of bitcoins is determined by the ability to provide a correct response to a cryptographic challenge. As miners gain block rewards only by mining valid blocks, they will take care to validate the ownership of the bitcoins being transferred in a regular transaction. They will also reject any transactions which attempt a double spend. The supply of new bitcoins is regulated by adapting the mining threshold to the total network hash rate. So the three challenges in building a decentralized virtual currency are addressed successfully by the Bitcoin protocol.

[11] Data source: https://blockchain.info/charts/total-bitcoins?timespan=all

2

The Security of the Bitcoin Protocol

The goal of the Bitcoin protocol is to operate a virtual currency system. We discuss its security from the perspective of a regular user who wishes to use the Bitcoin protocol *to store and exchange value*. For such a user, a *security failure* in the Bitcoin protocol manifests itself as either *a loss of some bitcoins* she owns or as *an inability to perform a Bitcoin transaction* in a timely manner.

We consider a scenario where a user stores her private keys (that can unlock her UTXOs) on a device in her control. The device could be a computer, a mobile phone, or a hardware wallet (which is a password-protected USB device for storing private keys). In this scenario, the user has the ability to create new transactions and transmit them on the Bitcoin network without revealing her private keys to a third party.

We do not consider the scenario where a user delegates the storage of her private keys to a web wallet or has a bitcoin balance in a Bitcoin exchange. In the latter case, the exchange holds the private keys for some UTXOs on the blockchain and maintains a traditional relational database with entries specifying the amount of bitcoin each of its customers has purchased from them. The combined balances of multiple customers may correspond to a single UTXO. While access to the web wallet or exchange may require multi-factor authentication from the user, these systems have attack surfaces similar to banking systems.

2.1 Bitcoin vs Banking Systems

Before we describe some possible attacks on Bitcoin, we compare it (albeit in a brief manner) to traditional electronic banking systems. This comparison is meant to clarify why certain common attacks on banking systems (like phishing) do not apply to the Bitcoin protocol. We also discuss some usability shortcomings of the Bitcoin protocol as compared to traditional banking and payment systems.

Network architectures Traditional electronic banking systems are based on a client-server network architecture while Bitcoin is based on a peer-to-peer network architecture. Bank customers connect to bank servers using clients (web browsers or mobile applications) and execute transactions which result in updates to the bank's private database. Bank servers, which host the databases and applications implementing the business logic, represent single points of failure. While banks have backup servers to guard against server hardware failure, an attacker who gains access to the server can create fraudulent transactions affecting several customers. Such an attacker is not necessarily an agent external to the banking system and could be a bank employee. In Bitcoin, if an attacker gains access to a node in the peer-to-peer network he can, in the worst case, steal the private keys of the node owner. The private keys of the other node owners are not compromised.

Bank servers are also vulnerable to distributed denial-of-service (DDoS) attacks where an attacker bombards the servers with a large number of packets from hacked computers. As the packets arrive from thousands of compromised computers, it is difficult to distinguish between the attack packets and those from genuine customers. The servers slow down or crash, resulting in service requests from genuine customers being delayed or dropped. Attackers demand ransom (typically in bitcoins) from banks with a threat of a DDoS attack at a particular time if their demands are not met. To perform a DDoS attack on the Bitcoin peer-to-peer network, an attacker has to simultaneously launch an attack on many nodes. This requires more resources from the attacks measured in terms of infected computers to be effective. Even if an attacker is able to mobilize the resources needed, a user whose node is under attack can stop it by changing her node's IP address (for example, by switching from a home Internet connection to a mobile hotspot connection). Bank servers do not have this flexibility as they need to maintain a consistent list of IP addresses.

Phishing and DNS Hijacking In Bitcoin, a user can create a new transaction on her personal computer. The private key used to unlock UTXOs never leaves the computer and only the transaction containing the response script is sent on the network. Furthermore, the transaction is protected by a digital signature which will become invalid if the transaction is tampered with. On the other hand, a bank customer who wishes to perform an online transaction needs to enter his password in a webpage. A common method used by attackers to steal customer passwords is to create a clone of the bank's login webpage and send the link to the customer in a phishing email.

Banking websites are also vulnerable to domain name service (DNS) hijacking attacks.[1] While a successful phishing attack requires that a bank customer clicks on a link which sends him to the attacker's clone of the banking website, DNS hijacking sends the customer to the malicious website even if he types the bank website URL correctly in a browser. This is made possible by the attacker taking control of some DNS servers on the Internet which perform the task of translating URLs to IP addresses. DNS hijacking attacks last for a few hours before they are discovered. Any customer who visits the bank website during the attack and enters his credentials would have revealed his username and password to the attacker.

Bitcoin also relies on DNS for the distribution of its software implementations. For instance, the Bitcoin Core implementations are available for download from https://bitcoin.org/en/download. If an attacker were to redirect users visiting this site to a malicious site, he can induce the users to download and install modified versions of the Bitcoin Core client. The attacker's version of the clients could be programmed to send the private keys of the user to the attacker. To avoid such attacks, users are encouraged to verify the authenticity of the downloaded clients by checking for the presence of the correct digital signatures.[2] This type of mitigation against DNS hijacking is not available for banking systems as regular bank customers do not have access to the bank website implementations.

Trust Requirements While banks have internal controls for fraud prevention by both criminals and their own employees, there is no foolproof way to ensure that these controls were correctly implemented for all transactions. While audit trails are created for every transaction in the banking system, bank system administrators can modify these trails if they wanted to. Sometimes dishonest

[1] https://en.wikipedia.org/wiki/DNS_hijacking.

[2] The recommended verification procedure is available at https://bitcoin.org/en/alert/2016-08-17-binary-safety.

employees may exploit loopholes in the auditing procedures to defraud customers.[3] The customer has to ultimately trust the competence and integrity of the bank employees to prevent, detect, and recover from fraudulent activity.

To perform transactions, a user needs to be able to send transactions to a miner node which will include them in its candidate blocks. It is possible that miner nodes exclude or censor transactions from some users in spite of such transactions offering transaction fees. For example, a merchant's competitor may bribe miner node operators to exclude transactions which pay the merchant's Bitcoin address. The competitor may stand to gain from the resulting disruption to the merchant's business. One mitigation of such an attack is to have merchants privately give customers a new Bitcoin address for every transaction, so that a miner node does not know which transactions to exclude from its candidate blocks.

To guard against double spending attacks, in addition to waiting for confirmations on the payment transactions, a merchant's node needs to connect to at least one honest peer. If all the peers of a merchant's node are malicious, they may misrepresent a chain which was deleted during a fork resolution as the surviving chain. This is called an *eclipse attack*, as the true state of the blockchain is eclipsed from the merchant by the malicious nodes.

Usability Problems in Bitcoin From a usability perspective, the main downside of Bitcoin is that there is no customer support or protection (which is not surprising as the participants in the Bitcoin protocol are not exactly customers of anyone). If a user loses her private keys due to theft or computer crash, the corresponding bitcoins are lost forever. On the other hand, if a user forgets his online banking password he can get the bank to reset it for him. If a user sends some bitcoins to an online retailer for some goods and the retailer does not ship the goods to the user, there is no way to execute a chargeback. Banks and credit card companies have dedicated staff to investigate and resolve disputes raised by their customers.

Another disadvantage of the Bitcoin protocol is that all the transactions are public. The amounts and destination addresses of the bitcoins involved in every transaction are not encrypted and can be read by anyone with access to the blockchain.[4] While Bitcoin addresses look like a string of random alphanumeric characters and do not contain personal information of their owners, it is possible to link the identity of real people to these addresses by correlating them with information available elsewhere. For example, merchants who accept bitcoins as payment may advertise their Bitcoin address in their store or website. Anyone can query the blockchain for the amount of bitcoins stored in such addresses. If the amounts are large, the merchants may be targeted by criminals who may attempt to take control of the merchants' private keys through hacking or the threat of physical harm. To get the account balance of a bank customer, criminals would have to either hack into the bank's network or corrupt/coerce bank employees. This is a higher barrier than querying the blockchain for the number of bitcoins received by a merchant's Bitcoin address.

Finally, the transaction throughput of the Bitcoin protocol is orders of magnitude lower than traditional payment systems. A block in the blockchain can have a size of at most 4 megabytes and each transaction occupies a few hundred bytes. As the inter-block time is approximately 10 minutes, the transaction throughput is limited to less than 1.8 million transactions per day or about 20 transactions per second.[5] In comparison, Visa can handle 65,000 transactions per second which is three orders of magnitude higher than Bitcoin.[6]

[3] For details and case studies see *Chapter 10: Banking and Bookkeeping* in Ross J. Anderson, *Security Engineering: A guide to building dependable distributed systems*, Second edition, John Wiley & Sons, 2008.

[4] Websites like https://blockchain.info provide free read access to the blockchain.

[5] The assumptions used to get these estimates are available at https://bitcoin.stackexchange.com/q/59408/56894.

[6] Source: https://usa.visa.com/content/dam/VCOM/global/about-visa/documents/visa-facts-figures-jan-2017.pdf.

2.2 Preventing Theft

As mentioned before, ownership of bitcoins is determined by the ability to generate response scripts which correctly answer the challenge scripts in UTXOs. The challenge scripts are generated from Bitcoin public keys while the response scripts are generated from Bitcoin private keys. Stealing bitcoins owned by a user is equivalent to getting access to her private keys. As per our assumption in the beginning of this chapter, the private keys are stored on a device in the user's control. An attacker can steal the private keys by hacking into the user's device or by old-fashioned extortion. A user must guard against such attacks by keeping the devices storing her private keys free of malware and out of the reach of thieves and hackers. The best defense, however, is for a user to conceal the very fact that she owns some bitcoins.

The Bitcoin protocol provides a type of challenge script called *m-of-n multisignature (multisig) script*, which can be used to make hacking and extortion attacks more difficult to execute. An m-of-n multisig challenge script contains n public keys. The corresponding response script can be generated using *any m out of n private keys* that were used to generate the n public keys. For example, a 2-of-3 multisig response script can be generated using any two of the three private keys related to the three public keys. If the three private keys are stored on three different devices, a hacker would have to break into two of these devices to be able steal the bitcoins secured by the 2-of-3 multisig challenge script. While a 3-of-3 multisig challenge script would increase the security, it is less robust against accidental loss of private keys. If any one of the three private keys required for unlocking a 3-of-3 multisig script is lost due to a device crash, the UTXO becomes unspendable. On the other hand, a UTXO secured by a 2-of-3 multisig challenge script remains spendable even if one of the private keys is lost.

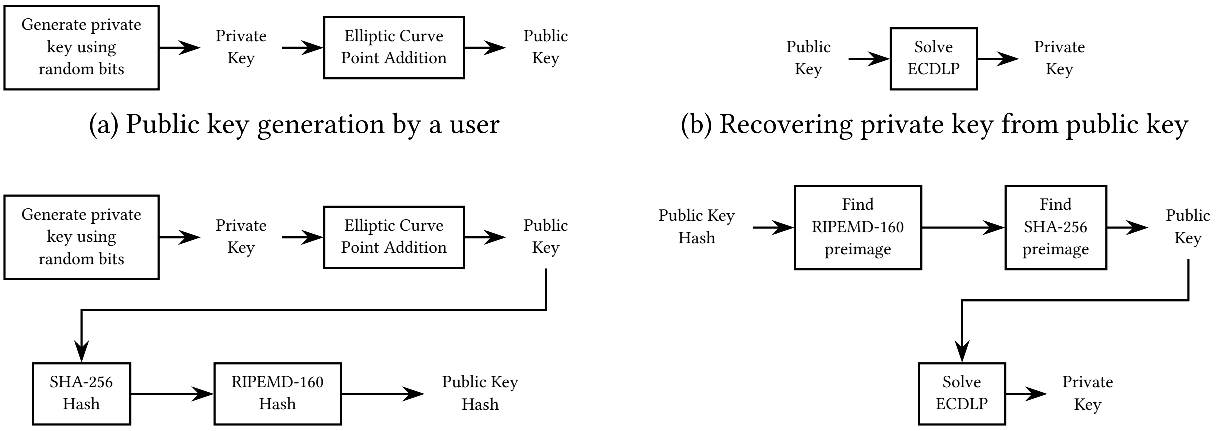
Let us turn our attention to potential ways of stealing private keys using only the information stored in the blockchain and see how the Bitcoin protocol avoids such attacks. Bitcoin uses a popular form of asymmetric cryptography called *elliptic curve cryptography*. In the context of cryptography, an elliptic curve is a set of points on a discrete two-dimensional plane satisfying a cubic equation in two variables. Each elliptic curve has an "addition" operation associated with it which takes two points P_1 and P_2 on the curve and returns another point on the curve which is denoted by $P_1 + P_2$.

Bitcoin uses a specific elliptic curve called `secp256k1`. The number of points on this curve (denoted by n) is a 256-bit integer which is more than 10^{77}. In Bitcoin, the private keys are just integers in the range $1, 2, 3, \ldots, n-1$. The public keys are obtained from the private keys by an algorithm called *elliptic curve point addition*. The `secp256k1` specification includes a special point P called the *base point*. The public key corresponding to a private key k is obtained by adding the base point to itself k times, i.e.

$$\text{Public Key} = \underbrace{P + P + \cdots + P + P}_{k \text{ times}}.$$

The overall procedure employed by the user to generate the public key is shown in Figure 2.1a. The first step is to use some random bits to generate the integer k which is the private key. It is important for the user to utilize an unpredictable source of randomness (like mouse movements on a screen) to defend against an attacker guessing the private key directly.

For convenience, the public key corresponding to a particular k is denoted by kP. The relationship between the private key k and the public key kP is one-to-one, i.e. there is exactly one public key corresponding to a given private key. Since the challenge scripts are derived from the public keys, the attacker sometimes has access to the public key. One example is the multisig challenge script and another example is a type of challenge script called *pay to public key (P2PK)*, which simply consists of a single public key. If the attacker could somehow compute k from kP, he would have recovered



(a) Public key generation by a user

(b) Recovering private key from public key

(c) Public key hash generation by a user

(d) Recovering private key from public key hash

Figure 2.1: Achieving consensus after a blockchain fork

the private key. The problem of finding k from the public key kP is called the *elliptic curve discrete logarithm problem (ECDLP).*

Note that the ECDLP cannot be efficiently solved by a brute-force search for k, i.e. by calculating iP for each i value in the sequence $1, 2, \ldots, n - 1$ and checking if $iP = kP$. It would take too long to find large values of k. For instance, even if we could check a million i values in one nanosecond it would take more than 300 years to find values of k larger than 10^{25}. As the number of private keys is more than 10^{77}, the probability that a randomly chosen private key is larger than 10^{25} is approximately $\left(10^{77} - 10^{25}\right)/10^{77} = 1 - 10^{-52}$ (which is a probability very close to 1).

The ECDLP has been extensively studied by mathematicians and cryptographers for decades. The best known algorithms require about 10^{38} operations to solve it for `secp256k1`, which is out of reach of the computing power available today and the far future. We cannot say that efficient algorithms for solving the ECDLP will never be discovered. It is possible that efficient algorithms may one day be discovered through some breakthrough in the theory of elliptic curves. In fact, the ECDLP may have become more attractive as a research area due to the billions of dollars worth of bitcoin it secures by being a difficult problem. But it will be difficult for an attacker to become rich by solving the ECDLP. Recall that the value in bitcoins can only be realized by exchanging them for goods or fiat currency. If an efficient algorithm for the ECDLP were to become public, the value of a bitcoin would plummet. To avoid this scenario, an attacker who discovers the solution to ECDLP would have to keep his discovery a secret and disguise the theft of private keys using his algorithm as regular hacking. Even the suspicion that someone has discovered a solution to the ECDLP for the `secp256k1` curve is enough to cause the value of a bitcoin to plummet, and the attacker's efforts to benefit from his discovery will be in vain.

To guard against a future scenario where an efficient solution to the ECDLP is found, the Bitcoin protocol provides a method to hide the public keys using cryptographic hash functions. For example, instead of using the P2PK challenge script to secure a UTXO another type of challenge script called *pay to public key hash (P2PKH)* can be used. In this script, the cryptographic hash of the public key is stored instead of the public key itself. In fact, the public key is sequentially hashed by two different cryptographic hash functions SHA-256 and RIPEMD-160 (as shown in Figure 2.1c). As cryptographic hash functions are difficult to invert, this increases the security of the bitcoins stored in a UTXO with a P2PKH challenge script. An attacker with access to only the public key hash would have to invert (find preimage of) both RIPEMD-160 and SHA-256 outputs before he can attempt a solution of the ECDLP on the public key (as shown in Figure 2.1d).

The public keys in a multisig challenge script can be hidden by using a type of challenge script called *pay to script hash (P2SH) multisig*. When a response script corresponding to a P2PKH or P2SH multisig challenge script is generated, it is required to contain the public keys. So the public keys are revealed when a UTXO secured by a hash-based challenge script is spent. If the same hash-based challenge script is used to secure multiple UTXOs, the extra protection provided by the hash functions is lost when one of the UTXOs is spent. So challenge script reuse (or equivalently Bitcoin address reuse) should be avoided.

2.3 Preventing Transaction Tampering

Suppose Alice wants to transfer some bitcoins to Bob. As illustrated in Figure 1.4, she creates a new transaction with inputs containing response scripts, which unlock some of her UTXOs already present in the blockchain, and outputs containing challenge scripts only Bob can unlock. Alice will transmit this new transaction to her peers in the Bitcoin network, who will then forward it to their peers. The transaction will eventually reach a miner node who will add it to the blockchain via the mining process. One might wonder how the following malicious behaviours are prevented:

1. A node who is a peer of Alice may modify the transaction sent by Alice before it reaches a miner node. The peer node can attempt replacing the Bob's challenge scripts in the transaction outputs with its own challenge scripts.

2. If Bob could modify the amounts in the transaction outputs, he could gain more bitcoins than what Alice intended to transfer to him. For example, suppose Alice creates a transaction which unlocks her UTXO containing 3 bitcoins and has two outputs — the first one sending 1 bitcoin to Bob and the second one sending 1.9999 bitcoins back to Alice as change (we assume a 0.0001 transaction fee). If Bob can change the amounts, he can pay himself 2 bitcoins in the first output and send only 0.9999 bitcoins back to Alice.

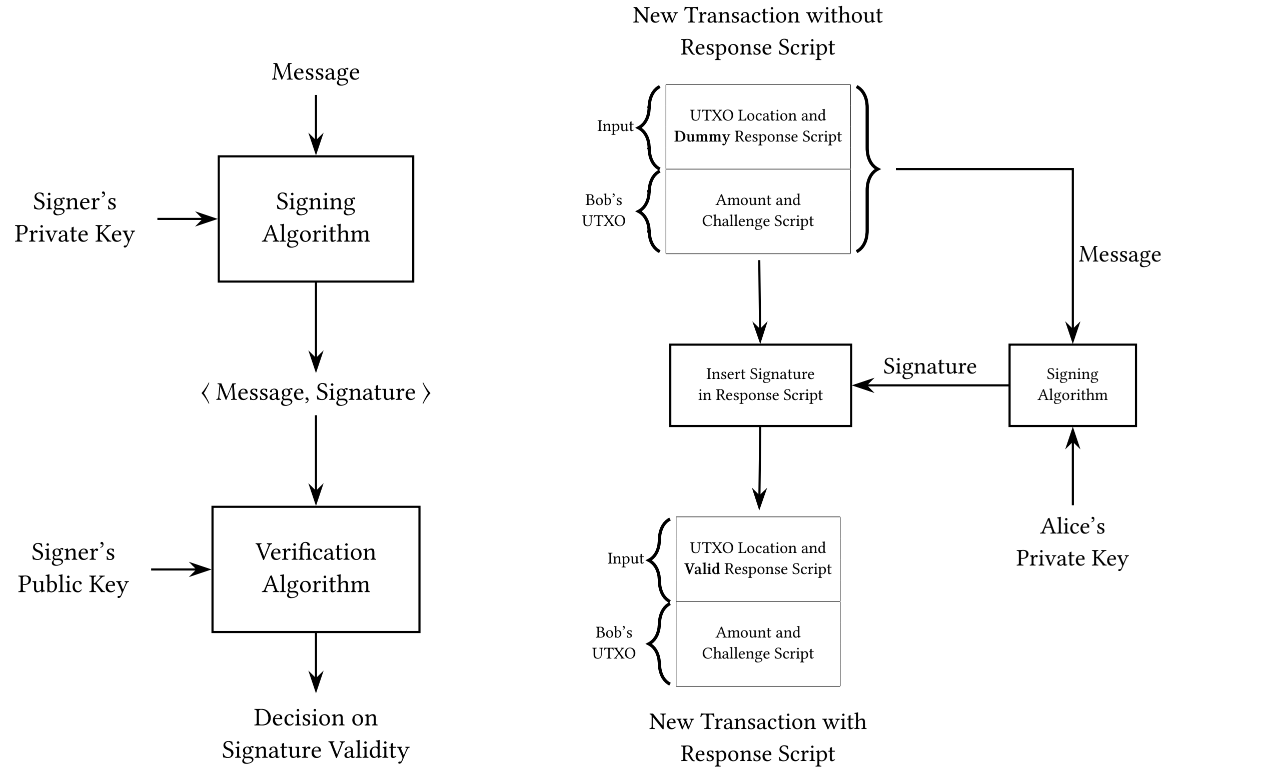
3. If Bob could change the output location specified in the transaction input, he could launch replay attacks. For example, Bob could use the modified transaction to unlock other UTXOs containing bitcoins which Alice did not intend to give Bob.

4. If Alice herself could change the transaction after receiving some goods or services from Bob, she can try modifying the amount of bitcoin being transferred to Bob. Or she can try to delete the transaction itself from the blockchain which has the effect of changing the state of the UTXO she used in the transaction input from spent to unspent.

The first three attacks are prevented using digital signatures where the ignorance of the private key prevents attackers from modifying the transaction created by Alice. In the fourth attack, Alice herself is the attacker and she knows the private key. Alice can modify an unconfirmed transaction (i.e. a transaction which has not appeared on the blockchain) by regenerating a valid digital signature for the modified transaction. But once a transaction has received enough confirmations, Alice cannot modify or delete it unless she controls a majority of the total network hash rate which is mining blocks. We discuss the mechanisms which prevent tampering of unconfirmed and confirmed transactions in the following two subsections.

2.3.1 Unconfirmed Transactions

Figure 2.2a illustrates the general workflow for using digital signatures to ensure that some message data has not been modified after it has been signed. The signing algorithms generates a signature


Message
Signer's Private Key
Signing Algorithm
⟨ Message, Signature ⟩
Signer's Public Key
Verification Algorithm
Decision on Signature Validity
New Transaction without Response Script
Input
UTXO Location and Dummy Response Script
Bob's UTXO
Amount and Challenge Script
Message
Insert Signature in Response Script
Signature
Signing Algorithm
Alice's Private Key
Input
UTXO Location and Valid Response Script
Bob's UTXO
Amount and Challenge Script
New Transaction with Response Script

(a) Workflow of digital signature generation and verification in general

(b) Digital signature generation and insertion into response script in Bitcoin

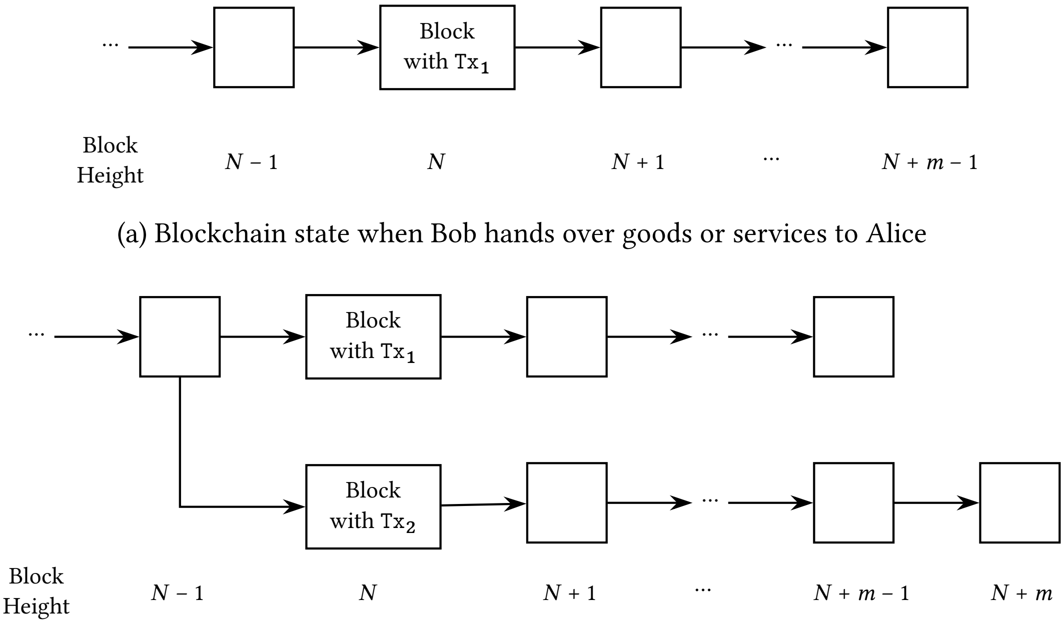
Figure 2.2: Digital signatures in general and in Bitcoin

for a given message using the signer's private key. Both the message and signature are required by the verification algorithm in addition to the signer's public key. The verification algorithm outputs a decision on the validity of the signature. Unlike handwritten signatures, a digital signature depends on the message. Modifying even a single bit in the message or the signature will render the signature invalid.

Suppose Alice wants to transfer some bitcoins to Bob. She creates a new transaction which has an input specifying the location of the UTXO that will be used to fund the transfer and an output specifying the amount and Bob's UTXO. Initially the response script in the input will be empty. Alice inserts a dummy response script in the input and signs the transaction using her private key as shown in Figure 2.2b. The signature is then inserted into a valid response script which replaces the dummy response script in the new transaction. Anyone who receives this transaction can verify the signature by reconstructing the message (replacing valid response script with dummy response script) and reading Alice's public key either from the challenge script in her UTXO or from the response script.[7]

As the output of the transaction is signed, it cannot be modified by a peer node who does not know Alice's private key. If the peer attempts to sign the transaction using its own private key, the resulting response script will not be a valid response to unlock Alice's UTXO. For the same reason, Bob cannot change the amounts in the output. Bob cannot reuse the same response script to unlock Alice's other UTXOs because the UTXO location was part of the message which was signed. Changing the UTXO location will change the message and the signature will not pass the verification algorithm.

[7] Alice's public key is available in the UTXO in case of non-hash-based challenge scripts (like P2PK or multisig) and in the response script in case of hash-based challenge scripts (P2PKH or P2SH multisig).



(a) Blockchain state when Bob hands over goods or services to Alice

(b) Blockchain state after Alice succeeds in creating a longer chain containing $\mathtt{Tx}_2$

Figure 2.3: Tampering of confirmed transactions

2.3.2 Confirmed Transactions

In the scenario described in the previous subsection, Alice herself may want to modify her transaction to cancel her transfer of bitcoins to Bob. As she has the private key which generates valid signatures, she can always re-sign the modified transaction. To avoid transaction tampering by Alice, Bob typically waits for a few transaction confirmations before providing goods or services in exchange for the bitcoins. Suppose the original transaction where Alice transferred bitcoins to Bob was $\mathtt{Tx}_1$ and it was added to the blockchain in the block at height N. Figure 2.3a illustrates the state of the blockchain when Bob hands over goods or services to Alice after waiting for $\mathtt{Tx}_1$ to receive m confirmations.

To modify $\mathtt{Tx}_1$, Alice would have to change it in all copies of the blockchain stored by the network nodes. Changing just her local copy of the blockchain is not enough. The only way Alice can replace $\mathtt{Tx}_1$ with a modified transaction $\mathtt{Tx}_2$ is to create a blockchain fork which upon resolution deletes the block containing $\mathtt{Tx}_1$ and adds a block containing $\mathtt{Tx}_2$. Figure 2.3b illustrates the state of the blockchain after Alice has succeeded in constructing a chain containing $\mathtt{Tx}_2$ which is longer than the original chain. As the network nodes always switch to the chain which required the most work to produce, the chain containing $\mathtt{Tx}_1$ will be abandoned.

Note that Alice cannot reuse the proof-of-work from the original chain and has to mine all the blocks in the new chain. Once the transaction $\mathtt{Tx}_1$ is replaced with $\mathtt{Tx}_2$ in the block at height N, the field containing the cryptographic hash of the transactions in the block header changes. This is due to the *collision-resistance* property of cryptographic hash functions which give different hash outputs for different hash inputs. A change in the block header represents a change in the Hashcash-like challenge to the miner (who is Alice in this case). She has to repeat the process of finding the proof-of-work nonce which makes the block header hash of block N fall below the threshold. Change in the block N header also results in a change in the Hashcash-like challenge posed to the miner in block $N + 1$, as every block header contains the hash of the previous block header. The changes in the block headers cascade and Alice has to re-mine all the blocks at height $N + 1, N + 2$, and so on.[8] This requirement on an attacker attempting modification of confirmed transactions is precisely the reason for including the cryptographic hash of the previous block header in the current block header.

While Alice is mining on the chain containing $\texttt{Tx}_2$, the other honest miners will continue to mine on the chain containing $\texttt{Tx}_1$, adding new blocks to this chain. Irrespective of the number of confirmations Bob waits for, Alice will succeed in constructing the longer chain if she can influence 50% or more of the total mining hash rate to mine blocks on the chain containing $\texttt{Tx}_2$. An attacker with such a capability is called a *51% attacker*. As of May 2017, the total mining hash rate working on Bitcoin is about 27 exahashes per second. The retail cost of a single mining device which can perform 14 terahashes per second is about 1200 USD.[9] As one exahash equals 10^{18} hashes and one terahash equals 10^{12} hashes, the total mining hardware working on Bitcoin can be visualized as consisting of 2 million such devices. While wholesale costs may be lower, the total retail cost of these devices is about 2.4 billion USD. So a 51% attacker has to influence the direction of equipment which costs at least 1.2 billion USD. This calculation ignores the capital costs of the data centre infrastructure and the operating costs of electricity and maintenance. It suffices to say that a 51% attack can only be launched by a wealthy attacker.

The Bitcoin protocol does not provide any cryptographic protection against a 51% attacker. This attack is not seen in the wild for economic reasons. If some miners controlling 50% or more of the total mining hash rate were to collude to tamper with confirmed transactions, there would be a general loss of confidence in the utility of Bitcoin as a payment system. This would cause the value of a bitcoin to fall, reducing the value of the block rewards earned by the miners. Given the significant investments made by miners to deploy their mining infrastructure, they would not want to preserve the status quo and not jeopardize the value of their future block rewards.

2.4 Security Against Protocol Disruption

In this section, we consider attackers who do not aim to gain bitcoins but want to disrupt the operation of the Bitcoin protocol by preventing users from performing transactions in a timely manner.

2.4.1 Mining Empty Blocks

For a transaction to be confirmed, it needs to be included in a block by a miner. If miners disregard the transaction fees offered by regular transactions and begin mining empty blocks (blocks containing only the coinbase transaction), they will still receive the block subsidy consisting of the newly created bitcoins. If all miners were to mine empty blocks, no user would be able to perform a Bitcoin transaction. Even if some of the miners were to mine empty blocks, the transaction backlog would increase and lead to an increase in the transaction fees. The latter is because miners mining nonempty blocks would prioritize transactions paying higher fees for inclusion in the blockchain. So users who want their transactions to be confirmed quickly would have to pay a higher fees in order to move their transactions to the head of the transaction queue.

Miners mining empty blocks may cause Bitcoin to be perceived as an inefficient and unreliable payment system. This could lead to reduction in the demand for bitcoins and a corresponding fall in their value. In addition to the loss of transaction fees from regular transactions, the empty block miners stand to lose money due to the fall in the exchange rate.

[8] The same argument holds if instead of replacing $\texttt{Tx}_1$ the attacker tries to delete it from the block. Removing $\texttt{Tx}_1$ from the list of transactions modifies the cryptographic hash of the transactions in the block header. The difference is that modifying $\texttt{Tx}_1$ requires knowledge of a private key while deletion does not require it.

[9] We consider the Antminer S9 manufactured by Bitmain https://shop.bitmain.com/.

2.4.2 Spam Transactions

An attacker can increase the transaction confirmation delays experienced by a user by transmitting a lot of his own transactions which pay a higher fee. These spam transactions may not correspond to any business process and may be a simple transfer between two addresses owned by the attacker. As the attacker's transactions get confirmed and occupy space in new blocks, a regular user would have to tolerate delays or pay a higher fee. This type of attack is not generally feasible in the long run because of the transaction fees being paid by the attacker to the miners. One scenario where this attack is feasible is when the miner himself is the attacker. In this case, the transaction fees in the spam transactions contained in blocks mined by the attacker are routed back to him. But if other honest miners mine blocks containing the spam transactions, the attacker loses the bitcoins making up the transaction fees. So the attacker would have to control a significant portion of the mining hash rate to be able to launch a long term spam attack. As before, such an attacker would not want to jeopardize his investment in his mining infrastructure by reducing the utility of Bitcoin as a payment system.

2.5 Conclusion

Bitcoin has cryptographic protection mechanisms where possible and economic protection mechanisms for attacks not defendable by cryptography. The 51% attack represents the biggest threat to the security of the Bitcoin protocol as a payment system. But even this type of attacker can only add or delete transactions and cannot modify transactions or steal bitcoins without knowledge of the private keys. While launching a 51% attack requires significant expenditure with little financial returns, it is not out of reach of a hostile nation state. Until an adversary of that stature emerges, the Bitcoin protocol can be considered secure.

3

Further Reading

For a gentle introduction to Bitcoin and cryptocurrencies, the following book is a good starting point. A free pre-publication draft is available at http://bitcoinbook.cs.princeton.edu/. The site also has links to video lectures by the authors.
Arvind Narayanan, Joseph Bonneau, Edward Felten, Andrew Miller, and Steven Goldfeder. *Bitcoin and Cryptocurrency Technologies: A Comprehensive Introduction*. Princeton University Press, 2016.

For a broad and in-depth treatment of Bitcoin targeted towards engineers, see the following book. The content of the book is available for free download at https://github.com/bitcoinbook/bitcoinbook.
Andreas M. Antonopoulos, *Mastering Bitcoin: Programming the Open Blockchain*. 2nd Edition, O'Reilly Media, Inc., 2017.

For a brief and technical introduction to Bitcoin, see the following lecture notes written by me. These notes, while not as comprehensive as the above two books, cover topics like elliptic curve cryptography and script execution in considerable detail. The target audience for these notes are graduate students in computer science and electrical engineering. The notes can be downloaded from https://www.ee.iitb.ac.in/~sarva/bitcoin.html
Saravanan Vijayakumaran, *An Introduction to Bitcoin*, Version 0.1, October 2017.

About the Author

Saravanan Vijayakumaran received a BTech (ECE) from IIT Guwahati in 2001 and a PhD from the University of Florida, Gainesville in 2007. After a brief stint at Microsoft in Redmond, he joined the the Department of Electrical Engineering at IIT Bombay in 2009 where he is currently an Associate Professor. He is a recipient of the Dr. Shankar Dayal Sharma Gold Medal at IIT Guwahati in 2001, a University of Florida Alumni Fellowship for the years 2001–2005, and the Departmental Award for Excellence in Teaching at IIT Bombay in the year 2017.

His research interests include error correcting codes and signal processing for communications. In 2016, he became interested in cryptocurrency technologies after attending the Scaling Bitcoin Workshop held at Milan. He took a one-year sabbatical in 2017 to learn more about Bitcoin and write a book on it (first draft available at https://www.ee.iitb.ac.in/~sarva/bitcoin.html). He was an invited speaker at Blockchain 2017, a workshop organized by ISI Kolkata and Microsoft Research Labs, India. He gave a half-day tutorial on Bitcoin at the National Conference on Communications 2018 organized at IIT Hyderabad. He has taught two day-long courses on Bitcoin for working professionals as part of the Continuing Education Programme at IIT Bombay.