Bachelor's Thesis (UAS)

Degree Program in Information Technology

2012

Rostislav Skudnov

# BITCOIN CLIENTS

TURUN AMMATTIKORKEAKOULU
TURKU UNIVERSITY OF APPLIED SCIENCES

BACHELOR'S THESIS | ABSTRACT

TURKU UNIVERSITY OF APPLIED SCIENCES

Degree Programme in Information Technology

03.06.12 Pages: 32

Instructor: Raija Tuohi

Rostislav Skudnov

# BITCOIN CLIENTS

Bitcoin is a new decentralized electronic currency which gained popularity in the last two years. The usage of Bitcoin is facilitated by software commonly called Bitcoin clients. This thesis provides an overview of Bitcoin and cryptography behind it, discusses different types of Bitcoin clients and researches additional features implemented by them. It also analyzes further enhancements that can be made to clients and the Bitcoin protocol.

Bitcoin clients are grouped into types and analyzed from a usability and security perspective. Security is very important for Bitcoin clients as they are used to manipulate money, and poor security leads to direct loss of money. Various threats are evaluated, including malware infestations, theft of files, hostile takeover of servers and hardware failures. Security implications of additional features and future enhancements are also assessed.

Various client types rely on significantly different security assumptions. While some clients are immune to hostile takeover of servers, for other clients this results in theft of money. None of the current clients is able to resist malware effectively. Additional features usually increase either security or usability, though some features improve both.

The current choice of Bitcoin clients and their feature set is much richer than that one year ago. New versions with more features are released very often. One of the future enhancements, multi-signature transactions, significantly increases security as it protects the money even if a client is totally compromised.

KEYWORDS:

Bitcoin, peer-to-peer, cryptography, digital signatures, ECDSA, deterministic generation, QR codes

# FOREWORD

I would like to thank all the teachers in Turku University of Applied Sciences for their hard work, help and support. Particularly, I want to send by best regards to my supervisor Raija Tuohi whose lectures on probability, statistics and cryptology inspired me to research these topics further, and whose help in writing this thesis was invaluable.

It was a pleasure to discuss matters about Bitcoin on Bitcointalk forum and IRC channels. My special thanks goes to Slush, ThomasV, jgarzik and etotheipi, who shared their knowledge and experience with me and whose productivity is unimaginable!

04.06.2012 Turku

Rostislav Skudnov

# CONTENTS



**FIGURES**

**TABLES**

# ACRONYMS, ABBREVIATIONS AND SYMBOLS

API – Application Programming Interface

CPU – Central Processing Unit

DSA – Digital Signature Algorithm

ECC – Elliptic Curve Cryptography

ECDSA – Elliptic Curve Digital Signature Algorithm

DDoS – Distributed Denial of Service

GPU – Graphics Processing Unit

GUI – Graphical User Interface

QR code – Quick Response code

RSA – Rivest, Shamir, Adleman

UNIX – UNiplexed Information and Computing Service

UTC – Universal Coordinated Time

URI – Uniform Resource Identifier

URL – Uniform Resource Locator

# 1 Introduction

Until recently, most monetary systems in the world have relied on financial authorities to issue and maintain money. E-commerce relies on those authorities or other financial institutions to serve as trusted third parties when processing payments. Even though this works relatively well in most situations, reliance on trusted parties incurs certain drawbacks. Many financial institutions require users to disclose their identities. Merchants suffer from chargebacks and are forced to raise prices and ask customers for more personal information than normally needed for providing goods and services. As a result, consumers are left with little privacy. Besides that, financial institutions may “freeze” customers’ accounts or cause other inconvenience, since the customers depend on them heavily.

In 2008, a decentralized electronic currency called *Bitcoin* was proposed by Satoshi Nakamoto [1]. It became the first widely-adopted online currency that does not require the use of financial institutions to conduct transactions. At the time of writing (May 2012), a few hundreds of companies and individuals listed on the Trade page of the Bitcoin wiki [2] are accepting Bitcoin as a payment. In the last two years, the value of Bitcoin rose from less than 1 US cent [3] in July 2010 to 30 US dollars in June 2011, and gradually declined to 5 US dollars, where it remains now (May 2012) [4].

The use of Bitcoin is facilitated by software programs which are referred to as "Bitcoin clients". The objectives of this thesis are to categorize Bitcoin clients by type, compare them to each other and evaluate them from a usability and security point of view. Other goals are to explore the additional features and future enhancements of Bitcoin clients. Political, economical and legal issues that arise from the usage of Bitcoin are beyond the scope of this thesis.

Chapter 2 explains some cryptographic concepts, which are necessary to understand the Bitcoin protocol and the operation of Bitcoin clients. Chapter 3 provides an overview of Bitcoin core concepts and the Bitcoin protocol. In Chapter 4, Bitcoin clients are categorized by types and each type is discussed and evaluated. Chapter 5 discusses additional features that are implemented by Bitcoin clients. Chapter 6 explains enhancements to the Bitcoin protocol, which will be implemented in the future versions of Bitcoin clients. The conclusion summarizes the history of the Bitcoin project and the results of this research.

# 2 Cryptographic concepts

To understand the principles of Bitcoin, we also need to know certain cryptographic concepts. Cryptography is the art and science of keeping messages confidential and secure. In addition to that, cryptography is also able to provide *authentication*, *integrity* and *non-repudiation* of messages. In this context, these words have the following meanings:

- *Authentication*. It should be possible for the receiver of a message to ascertain its origin; an intruder should not be able to masquerade as someone else.
- *Integrity*. It should be possible for the receiver of a message to verify that it has not been modified in transit; an intruder should not be able to substitute a false message for a legitimate one.
- *Non-repudiation*. A sender should not be able to falsely deny later that he sent a message. [8]

Cryptography has multiple means of achieving the above-mentioned goals, and we describe some of them that are needed to understand Bitcoin.

If two parties want to send messages securely, they may use *encryption* to hide the actual contents of the messages (*plaintext*) and transform them to *ciphertext*, i. e., to make them unreadable by anyone else. The receiving party can perform *decryption* to recover the plaintext. Usually, the algorithms for encryption and decryption are well-known, and only the encryption/decryption *keys* are maintained secret. If both parties use the same *key* for encryption and decryption, they use a *symmetric encryption algorithm*. Symmetric algorithms alone provide confidentiality, but to achieve the other goals, we need other techniques, such as *hash functions*.

> A *hash function* is a function, mathematical or otherwise, that takes a variable-length input string (called a pre-image) and converts it to a fixed-length (generally smaller) output string (called a hash value). [8]

> A *one-way hash function* is a hash function that works in one direction: It is easy to compute a hash value from pre-image, but it is hard to generate a pre-image that hashes to a particular value. [8]

The output of a hash function is not dependent on input in any distinguishable way. These properties give us the possibility to use hash functions to verify integrity of messages: someone having the hash of a message can determine whether the

message is intact. This method is used, for example, in BitTorrent protocol: the .torrent file has hashes of the pieces of data, and the data is checked to verify that it has the same hashes after it is downloaded from the peers. If the hash of a downloaded piece of data does not match the one in the .torrent file, such piece is rejected and later downloaded from someone else [15]. As a result, peers are unable to send fake data and force the downloader to accept it.

When a digital message provides authentication, integrity and non-repudiation together, we say it has a *digital signature*, similar to a paper document with a handwritten signature [8]. Although it is possible to create digital signatures by using symmetric algorithms, hash functions and a trusted third party, as described in [8], this solution is inefficient.

*Public-key cryptography* introduces an absolutely new way of thinking about encryption, decryption and digital signing. In order to encrypt and decrypt messages, we create two different keys, or a *keypair*: the *public key* and the *private key*.

> It is computationally hard to deduce the private key from the public key. Anyone with the public key can encrypt a message but not decrypt it. Only the person with the private key can decrypt the message. [8]

If we give out our public key, anyone is able to send us messages encrypted with it, and those messages could not be read by anyone else than us. Public-key cryptography could also be used for digital signing: we can find the hash of the message and encrypt it with the private key, thus forming a digital signature. If someone who has the public key receives the message with the digital signature, it is possible for him/her to verify both the authenticity and integrity of the message by decrypting the signature with the public key and comparing the result to the hash of the message. The signed message also has the property of non-repudiation, that is, the sender is not able to falsely deny sending the message.

There are many public-key algorithms, and RSA algorithm is the most widely used. There is also another family of public-key algorithms, known as Elliptic Curve Cryptography (ECC). The discussion about algorithms themselves is beyond the scope of this thesis.

# 3 Bitcoin overview

## 3.1 Electronic Cash before Bitcoin

Since the introduction of public-key cryptography, various proposals have been made to make a monetary system based on it, such as David Chaum's "Blind signatures for untraceable payments" [5] and Ian Grigg's "Triple Entry Accounting" [6]. Chaum's paper suggests a system where payments are done anonymously and securely, though a trusted third party is still needed. David Chaum also founded DigiCash BV, the first company to provide a cryptographic digital currency. Even though DigiCash became rather well-known in the payment industry in the 1990's, the company went bankrupt in 1998 [7]. In Ian Grigg's paper, when two willing parties transact, the payer (Alice) creates a receipt, which includes the payer's and payee's (Bob) names, the amount of money to be sent, and the digital signature of the whole receipt, made with the payer's private key. This receipt, together with the current date and time, is signed by issuer of the money (Ivan). This is illustrated in Figure 1. From this moment, as Ian Grigg says, "The Receipt is the Transaction" [6], which means that we do not need to keep a whole history of all transactions, but only the latest receipts. Ian Grigg claims that this system was implemented for internal money in a company, and proved to be more efficient than old-style accounting [6].

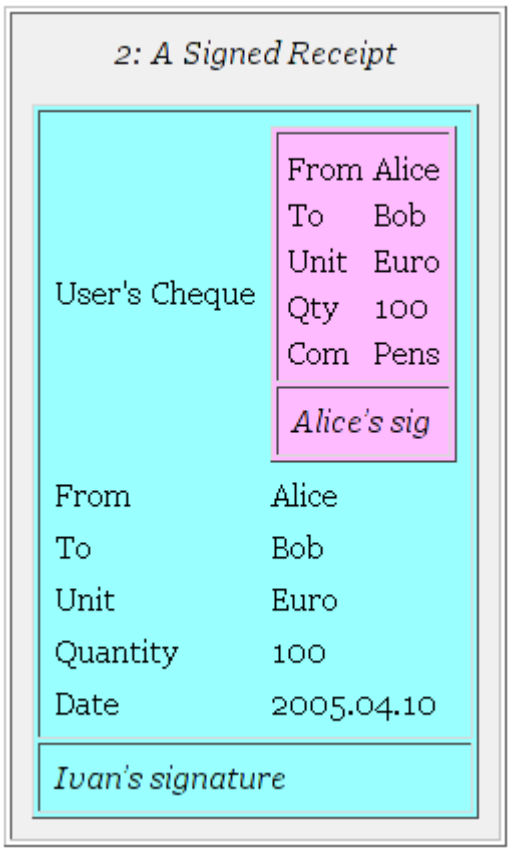


**Figure 1.** The Receipt is the Transaction [6]

The electronic currencies mentioned previously have the following common property: they are centralized, that is, they rely on a trusted party, the issuer, which facilitates and controls the transactions. In 2008, the first decentralized electronic currency called *Bitcoin* was proposed by someone named Satoshi Nakomoto [1]. The real identity of that person is not known.

## 3.2 Bitcoin basics

Bitcoin implements an accounting system similar to Ian Grigg's. We define an electronic coin as a chain of digital signatures, as shown in Figure 2.

> Each owner transfers the coin to the next by digitally signing a hash of the previous transaction and the public key of the next owner and adding these to the end of the coin. A payee can verify the signatures to verify the chain of ownership. [1]

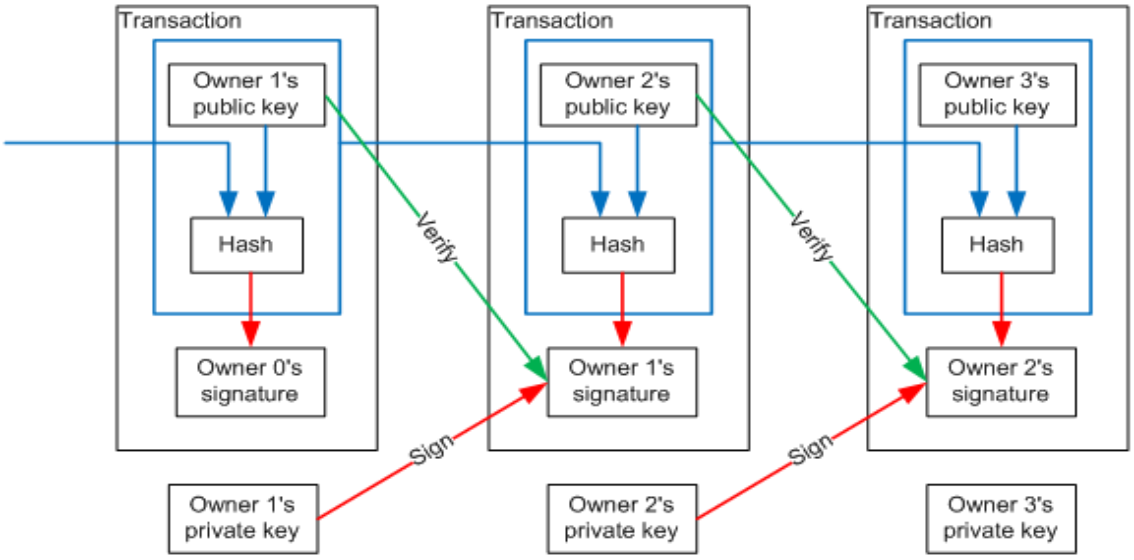


**Figure 2.** Coin as a chain of digital signatures [1]

In Ian Grigg's system, every transaction (receipt) is also signed by the trusted third party (issuer), which verifies that the payer has enough money to send and that the money being spent was not spent before (also known as *double-spending*). Bitcoin, on the other hand, makes all transactions public, so that everybody is aware of all transactions and is able to verify the chain of ownership and the non-existence of double-spending attempts. This idea as a theoretical concept was first described in Wei Dai's essay "B-money" [9].

Bitcoin relies on a *peer-to-peer overlay network*, built on top of the Internet, commonly referred to as *Bitcoin network*. This *peer-to-peer overlay network* is a special kind of network which differs a lot from how computer networks are usually constructed.

> Peer-to-peer systems are distributed systems consisting of interconnected nodes able to self-organize into network topologies with the purpose of sharing resources such as content, CPU cycles, storage and bandwidth, capable of adapting to failures and accommodating transient populations of nodes while maintaining acceptable connectivity and performance, without requiring the intermediation or support of a global centralized server or authority. [43]

*Overlay network* is “an application layer virtual or logical network in which end points are addressable and that provides connectivity, routing, and messaging between end points” [43]. Bitcoin network provides a communication channel to broadcast transactions and send other information between users (*nodes*) which is described later in this chapter.

As of May 2012, there are approximately 18000 nodes in the network even though the number varies with time [10]. Typically, Bitcoin nodes connect to 10-100 other nodes simultaneously.

Bitcoin's unique feature is the method for accepting/denying transactions and agreeing on a single history of transactions by the network. Due to propagation delays and connectivity issues, it is impossible to make everyone aware of all transactions at all times, and this can be abused by double-spending the money. Someone could spend the same money twice before the first transaction propagates far enough, so there must be a way to determine which transaction is valid. Figure 3 demonstrates how one coin, which belonged to Owner 1, can be spent to Owner 2 and Owner 3 at the same time.

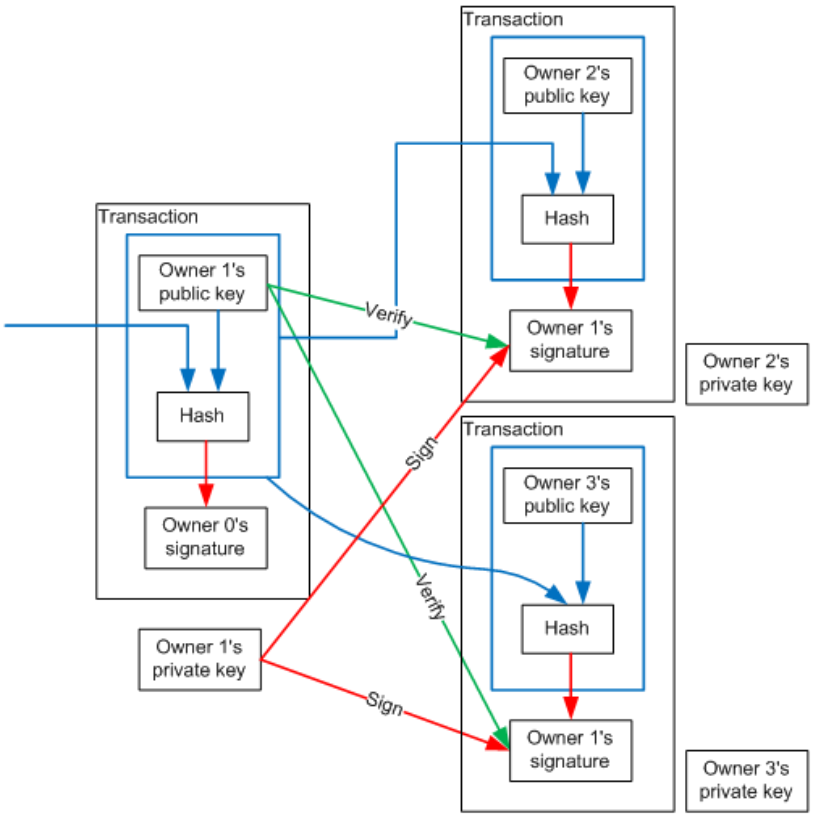


**Figure 3.** Double-spending

One obvious solution is to make those transactions that most people agree with valid. On the Internet people are represented by the software applications they are running and their respective IP addresses. If the validity of a transaction were determined by the majority of nodes, i. e. the majority of IP addresses, the system could be cheated by someone able to allocate many IPs. That is why Bitcoin uses a different way of

determining transaction validity. This technology is called *proof-of-work* and was originally suggested in Adam Back's Hashcash [11] as a measure to prevent email spam.

We provide a brief explanation of the principle of *proof-of-work*. For many cryptographic hash functions, the number of attempts to find an input whose hash begins with a certain substring can be probabilistically estimated, since the most efficient way to search for such an input is to brute-force by trying consecutive inputs [11]. An input that produces a hash with a certain leading substring is also called "partial hash collision", and the process of finding suitable input is called *mining* [12]. For example, if we need a certain 32-bit long leading substring in the binary representation of the hash, the expected number of inputs that we need to try is $2^{32}$, which is more than 4 billion. In Bitcoin, finding partial hash collisions serves as a proof that a certain amount of computation has been performed, this is why it is called *proof-of-work*.

Proof-of-work is used in Bitcoin for two purposes: the first is that proof-of-work is a means of “voting” about transaction history, where the more work one performs, the more voting power one has. The second purpose is the creation of money. Wei Dai writes:

> Anyone can create money by broadcasting the solution to a previously unsolved computational problem. The only conditions are that it must be easy to determine how much computing effort it took to solve the problem and the solution must otherwise have no value, either practical or intellectual. [9]

Finding partial hash collisions exactly fits this definition: it is easy to determine how much computing effort it took to find a given collision, and the collision does not have any other value. In Bitcoin, when a partial hash collision is found, it serves as a “vote” for certain transactions to be included in the history and provides a reward for the miner (participant engaged in Bitcoin mining).

Another important concept of Bitcoin is that every proof-of-work is based on some previous proof-of-work. This is implemented by including the hash from the previous proof-of-work into the input of the current proof-of-work, thus forming a chain, as shown in Figure 4. Input data for computing proof-of-work is combined into *blocks*, and all blocks together form a *block chain*.

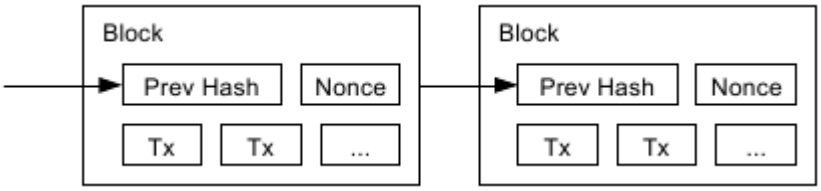


**Figure 4.** Chain of blocks [1]

As each block includes the hash of some previous block, we say that each block is built on top of some previous block and extends it. The “voting” happens when miners choose the block they wish to extend. Choosing a certain block implies agreement with all transactions in that block and all previous blocks relative to that block. If a group of miners works on a different block than others, the block chain may diverge, resulting in two or more competing sub-chains. The chain in which more computing power is invested will eventually become longer, and other Bitcoin nodes will prefer the longest sub-chain, discarding all other sub-chains [1]. As a result, the block chain includes only those transactions that the majority of processing power agrees with. In Figure 5, an example of chain divergence is shown, but the black chain is the longest and is preferred by Bitcoin nodes.

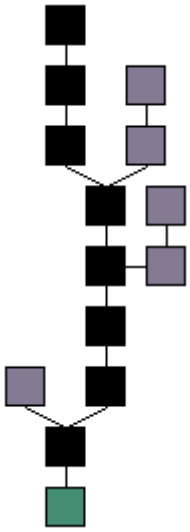

**Figure 5.** Block chain diverges, black chain wins

The *block chain* should not be confused with the coins themselves, which are chains of digital signatures. The block chain interconnects *blocks*, whereas chains of digital signatures interconnect *transactions*.

If a node follows the above rules, we consider it to be *honest*. An important condition which must be held for successful operation of Bitcoin network is that honest nodes

altogether always have more processing power than any attacker, and no attacker (or cooperating group of attackers) is able outperform all honest nodes together. While this condition is held, an honest block chain will always be longer than any of the attackers' chains, and will be preferred by other Bitcoin nodes [1].

## 3.3 Bitcoin implementation details

Though describing all implementation details of Bitcoin is beyond the scope of this thesis, some details should be examined to understand the features of Bitcoin clients.

Every block consists of a *block header* and the actual content, i. e. transactions. The block header contains the following information:

**Table 1.** Block header structure

| **Field** | Version | Previous hash | Merkle root | Timestamp | Bits | Nonce |
|---|---|---|---|---|---|---|
| **Bytes** | 4 | 32 | 32 | 4 | 4 | 4 |

*Version* is the same in all blocks

*Previous hash* – hash of the previous block header

*Merkle root* – hash which verifies the integrity of transactions in the block. The procedure for calculating Merkle root is explained later in this chapter.

*Timestamp* – time when block was generated, as a UNIX timestamp (number of seconds passed since 01.01.1970 00:00:00 UTC)

*Bits* – compact representation of the *target*, which designates the difficulty required for proof-of-work (1)

*Nonce* – value to be changed when mining in order to find partial hash collision.

Instead of storing transactions themselves in the block header, only the *Merkle root* is put there, which is the root hash of the *Merkle tree* computed from all the transactions to be included in the block. *Merkle tree* is generated by the following procedure. First, hashes of transactions are calculated. Then, these hashes are put pair wise and hashed again, producing a new, smaller set of hashes. This step is repeated multiple times until only one hash remains. Finally, this hash, which is called *root hash*, or *Merkle root*, is put into the block header. The exact procedure for calculating the Merkle tree can be found in the source code of Bitcoin clients [13]. As a result, every block

header has a fixed size of 80 bytes, and a possibility exists to verify transactions without having the full *block chain*, but only the headers. This process, known as "Simplified Payment Verification", is explained later, in the discussion about "headers-only clients".

Figure 6 illustrates the structure of the actual block #183301 of the Bitcoin block chain. Only five transactions are shown for simplicity, though the actual number of transactions is 432. Tx1 to Tx5 are transactions, H1 to H5 are hashes of transactions, Hash12, Hash34, Hash55 and Hash1234 are hashes of previous hashes. The previous block hash starts with 13 zeros in hexadecimal representation, which means 52 zeros in binary representation. Finding this proof-of-work requires $2^{52}$ attempts on average.

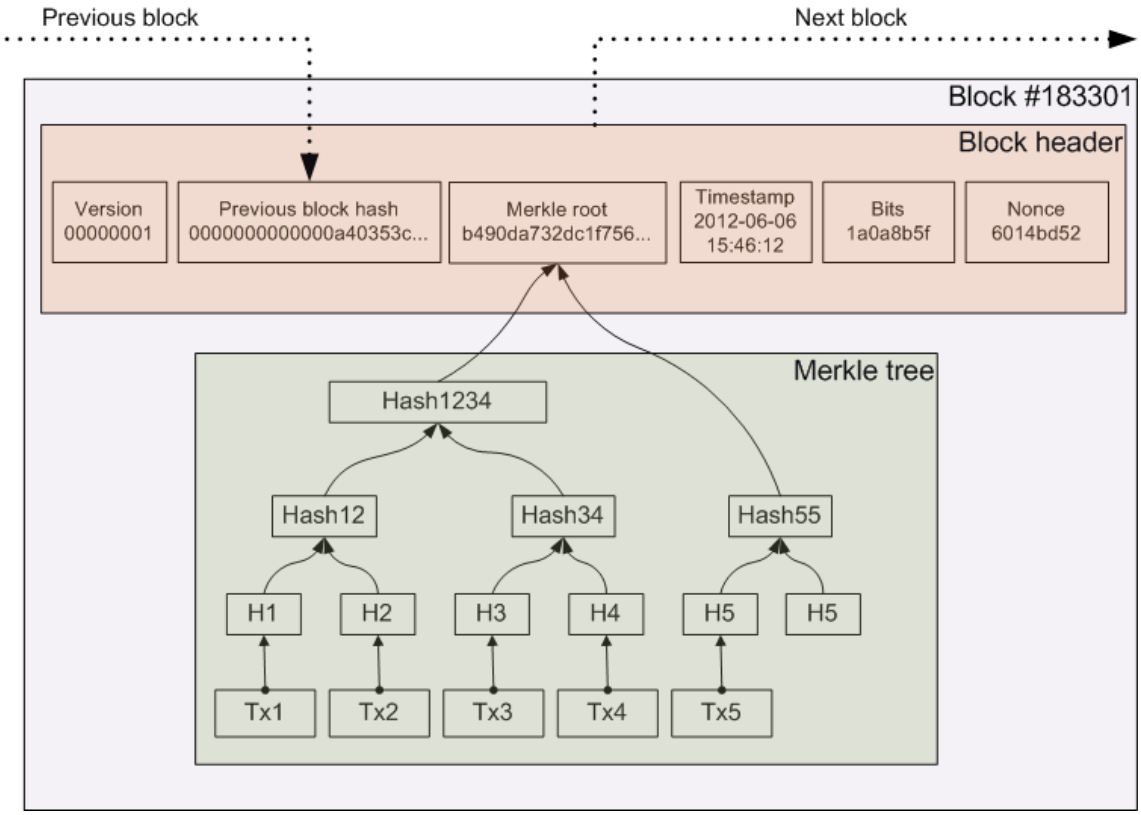


**Figure 6.** Block #183301 of the Bitcoin block chain

The transaction structure in Bitcoin is more sophisticated than that described in Section 3.1. Since it is impractical to transfer individual monetary units ("coins") separately, Bitcoin provides a way to split and merge "coins" in transactions. Each transaction has "inputs" and "outputs", where each output identifies the *address* of the receiver of coins and the amount received by him/her, and each input provides a reference to an earlier output that is being spent and a digital signature of the payer with the corresponding

public key. An *address* in Bitcoin is a hash of the address owner's public key. When verifying a transaction, the actual public key found in the *input* is hashed and compared to the *address* specified in the referenced earlier *output*. There are also "generation transactions", which give a reward to someone who finds a block, and these transactions have empty input. The digital signature algorithm used to sign transactions is ECDSA (Elliptic Curve Digital Signature Algorithm), which has several advantages over more widely used RSA/DSA: much smaller key size and faster computation while the security factor is the same [14].

# 4 Client types

## 4.1 Full clients

Full clients are the ones which implement the full Bitcoin protocol and hold a full copy of the *block chain*. This includes discovering and communicating with other nodes, sending and receiving transactions and blocks, saving all valid blocks locally, verifying all transactions received and broadcasting all legitimate transactions. In addition to those, full Bitcoin clients also provide services for the user. These services are: storing one's transaction history, private keys for the "wallet" and providing a Graphical User Interface (GUI), command-line interface or an Application Programming Interface (API) for viewing current balance, transaction history and initiating new transactions.

Some examples of full Bitcoin clients are: Original Bitcoin client [16], Armory [36], Libbitcoin [37].

We will look more closely at the operation of the Original Bitcoin client, also known as the Satoshi client. When Bitcoin was initially created, it was the only software that could be used for dealing with Bitcoin, hence the name. The Satoshi client keeps the *block chain*, nodes' addresses and the *wallet file* in the client's data folder. The wallet file contains the wallet owner's transaction history, address book and private keys, so it must be kept securely to prevent stealing of Bitcoins. Starting from version 0.4.0 the Satoshi client has wallet encryption feature, which is discussed further in Section 5.3. Since the early days of Bitcoin, and until version 0.3.22, the original client had *mining* capability [17], which was removed because specialized mining clients are much more efficient. Mining is further discussed in Section 4.5. Figure 7 shows the interface of the

Satoshi client. The overview screen has information about the current balance and most recent transactions. A transaction is said to be *unconfirmed* if it was sent but not included in a block yet.

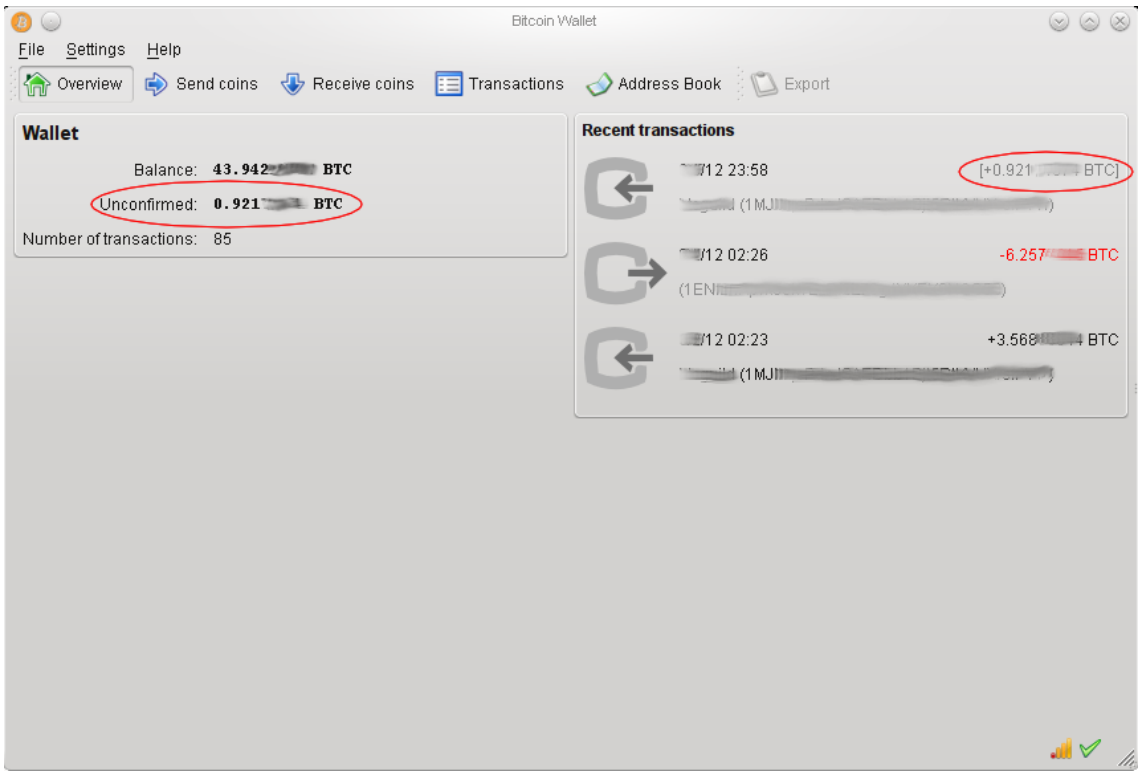


**Figure 7.** Unconfirmed transaction in original Bitcoin client, version 0.5.0

Using a full Bitcoin client has some drawbacks. One of them is excessive network and file system usage: a full client has to have a full copy of the *block chain* locally, which occupies 2 gigabytes as of May 2012 and which will only grow in the future. Full clients have to be aware of all transactions, so they receive and send transactions and blocks all the time and consume network bandwidth. On the other hand, operating a full client makes the Bitcoin network stronger and more difficult to attack.

## 4.2 Headers-only clients

For some users it may be difficult to store block chain data on their devices, for example, on mobile phones. Fortunately for such users, there are *headers-only clients* which don't require that much storage space. BitcoinJ software library [18] and clients based on it (e. g. Multibit [38]) do not download and store the full block chain but only the block headers, which occupy only 14 megabytes as of May 2012 and could be kept in memory even on mobile devices [18]. BitcoinJ downloads full blocks only sometimes,

when it expects incoming transactions and when it searches the block chain for keys that are in the wallet. Even though headers-only clients are not able to verify transactions against the full block chain, they are not less secure than full clients if some of these precautions are taken:

- Waiting for multiple blocks (usually 6) before considering the payment complete
- Receiving a copy of the transaction in question from a node trusted to be running a full Bitcoin client
- Receiving the transaction in question from multiple nodes [19]

BitcoinJ author Mike Hearn suggests a procedure for proving inclusion of a transaction in a block without having to download the whole block, but only the transaction itself and the corresponding Merkle branch [19]. Having this information is enough to verify that the transaction was included in a particular block. This procedure is known as Simplified Payment Verification and was also described in the original Bitcoin paper [1]. Using this procedure can save a lot of bandwidth and provide even greater security, but unfortunately, it has not been implemented in either BitcoinJ or any of the full clients.

### 4.3 Signing-only clients

The name of *signing-only* clients means that they only sign transactions, but do not deal with block chain or even block headers. Instead, these clients request data about certain transactions from the server. If a transaction happens in the Bitcoin network that involves one of the connected clients' wallets, the server may also push such information to a particular client that is interested in it. Signing-only clients send out only their own transactions. Consequently, the overhead of running such a client is much lower than that of full or headers-only clients, as the file system is used only for our own keys and transactions, and we only send and receive transactions that concern us.

Signing-only clients do not require much storage, network bandwidth and computing power, and therefore can be implemented in various ways: as a desktop application (Electrum [20]), mobile application (BitcoinSpinner [21]) or as a Web application (BlockChain.info [22]). In Web-based signing-only clients, cryptographic features are implemented in JavaScript and are executed in the Web browser, so the private keys of the wallet are never sent to servers unencrypted.

A certain level of trust is required in the server we are connecting to, because the server will know all our transaction history, and it is possible for the server operators to send us false transactions and trick us into thinking that we have more or less money than we actually do. This kind of attack is not dangerous and is not profitable for the server operators, because it is not possible for them to trick us into signing transactions against our will. In addition, this attack can be mitigated by connecting to multiple servers or to our own server, which can be set up with open-source software [20, 23].

Figure 8 demonstrates the process of sending Bitcoins with the Electrum client. The only required information is the “Pay to” (payee’s Bitcoin address) and the “Amount”. The transaction fee is calculated automatically.

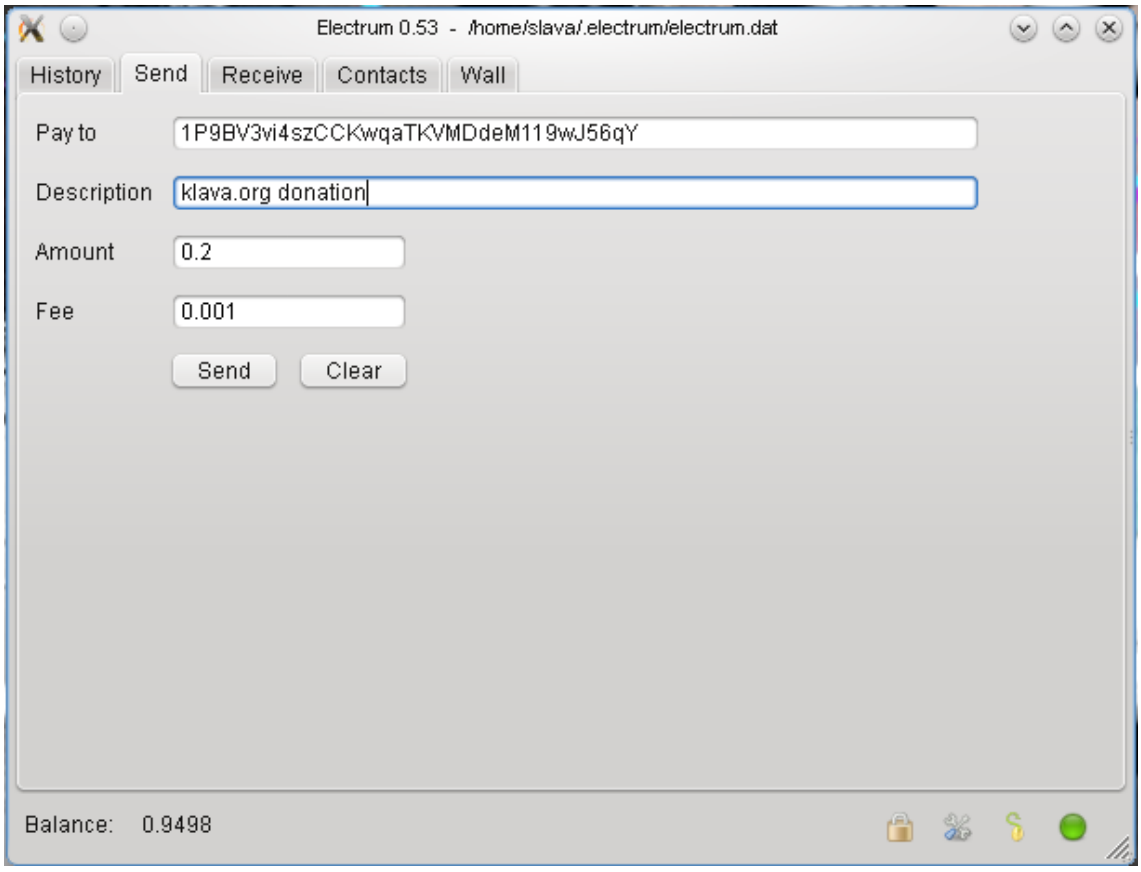


**Figure 8.** Transaction creation in the Electrum client

## 4.4 Thin clients

Thin Bitcoin clients (also called *eWallets* or browser-based wallets) are the ones which do not hold private keys and do not sign transactions themselves. Instead, they send commands to a remote server to perform these operations. The remote server acts in a

similar way to a bank, providing financial services to customers. The most important advantage of thin clients over other types of clients is that the user does not have to worry about backing up private keys and keeping them safe at the same time, as these operations are performed on server-side on behalf of thin clients. Another benefit is that thin clients are the easiest to set up: one only needs to visit a Web page of the eWallet and set up an account in one minute.

Some examples of thin Bitcoin clients are: MyBitcoin [24], Instawallet [25], MtGox wallet [26].

Instawallet does not even require registration: when visiting the page for the first time, a new account is created automatically [25]. Figure 9 is a screenshot of Instawallet's transaction creation dialogue, which is very simple and straightforward.

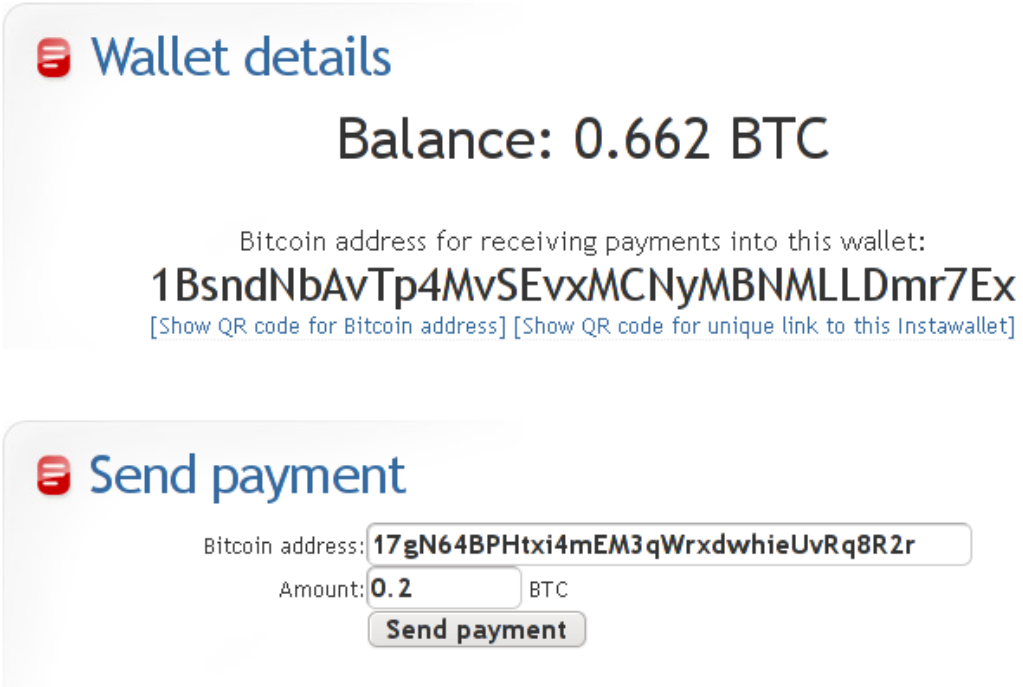


**Figure 9.** Transaction creation in the Instawallet client

The high usability of thin Bitcoin clients does not come without a price. Ultimate trust is required in the eWallet provider, since it not only knows the users' transaction history, but also has control over the users' money. There is no obvious way to ensure that the provider has the amount shown as the balance is backed by actual Bitcoins stored in reserve. Finally, the provider can become a victim of loss or theft of Bitcoins, or of a malicious takeover, which can result in a loss of customers' funds.

On July 29, 2011 MyBitcoin, the most popular eWallet provider at the time became inaccessible. After a week, it was announced that MyBitcoin was hacked and a half of the customers' funds were stolen by unknown persons [24]. This unfortunate event proves that one needs to choose wallet providers carefully or, preferably, not use thin clients at all.

In my opinion, thin Bitcoin clients do not have any advantage over existing banking infrastructure, where money is managed by trusted third parties. The main idea of Bitcoin is not to rely on them, but thin clients turn this idea down.

### 4.5 Mining clients

Bitcoin mining clients, or simply *miners*, are specialized clients that are not used to send or receive Bitcoins; their only usage is *mining*. Initially, the only mining client was the original Bitcoin client, which implemented mining on CPU. As more people started to know about Bitcoin and became involved in mining, at the end of 2010 the difficulty of finding blocks rose to such levels that it would take 1 year on average to generate a block and get the 50 bitcoins reward if mining is done on a single computer with the original client [28]. This was due to the fact that specialized mining clients were created to perform mining on graphics card's GPU, and these clients turned out to be 100 times more efficient than the original Bitcoin client, which still used CPU mining [27]. As a result, the built-in mining capability of the original client became obsolete and was removed in June 2011 [17]. GPU still remains to be the most popular mining hardware [27], and it can be used not only for mining, but also for playing games and using other software applications that require a lot of computational resources.

Some examples of Bitcoin mining clients that implement GPU mining are: Phoenix [39], CGMiner [40].

Even with a GPU, it took a few days on average to generate a block and get a reward. If someone was unlucky, he would not get a reward for weeks, because the rewards were too volatile. On 27$^{th}$ of November, 2010 forum member Slush suggested *pooled mining* (initially it was called *cooperative mining*) to combine the power of multiple miners to work on the same block [28]. Instead of getting a large reward once in a long time, miners started to get smaller rewards more frequently. Pooled mining became very popular and now accounts for more than a half of all Bitcoin mining [29].

Table 2 summarizes all client types and their features.

**Table 2.** Bitcoin client types and their features

| | Block chain | Block headers | Transaction signing | Mining |
|---|---|---|---|---|
| Full clients | ✓ | ✓ | ✓ | ✗ |
| Headers-only clients | ✗ | ✓ | ✓ | ✗ |
| Signing-only clients | ✗ | ✗ | ✓ | ✗ |
| Thin clients | ✗ | ✗ | ✗ | ✗ |
| Mining clients | ✗ | ✗ | ✗ | ✓ |

# 5 Additional features

## 5.1 Deterministic wallets

Some Bitcoin clients, including the Satoshi client and Multibit, generate cryptographic keys randomly. After initialization, the Satoshi client generates the wallet file with 100 keypairs in it [30], which correspond to 100 Bitcoin addresses. Since it is encouraged to use a different address for every payment, 100 addresses will be used up quickly. If that happens, the Satoshi client generates new keys when necessary. This way of operation makes backing up the wallet difficult: to ensure that all keys are safe, we need to do the backup after every transaction. If such backup is not done, a loss of the original wallet will result in a loss of Bitcoins belonging to the newly-generated address. The Satoshi client does not have any built-in backup functionality, and implementing frequent and secure backups is not an easy task for those not involved in IT. Fortunately, deterministic wallets provide a solution to this problem.

Some of the Bitcoin clients, for example, Electrum and Armory, generate keypairs and, consequently, addresses deterministically. Deterministic generation means that any

number of cryptographic keys is generated from a relatively small *seed*. A known deterministic algorithm produces a keypair from two arguments: seed and a sequence number. If the same seed and number are later supplied to that algorithm, it will always produce the same keypair [31]. As a result, when using a client with a deterministic wallet, we only need to backup the seed, and it has to be done only once. If we have to restore the wallet later, we will supply the seed to the client, and the client will restore all keys from the seed and the history of transactions from the block chain [20].

## 5.2 Brainwallet

The concept of *brainwallet* is closely related to deterministic wallets. At the current level of computing technology, randomly-generated numbers of 128 bits "can guarantee uniqueness across space and time" [32]. After generating a random number of 128 bits we can be sure that no one else in the universe is able to generate the same number independently. This number can be used as a seed for a deterministic wallet. At the same time, we can also convert this number to a human-readable form and memorize it. 128 bits can be represented as 128 zeros and ones, or as 32 hexadecimal characters, or as 24 characters in Base64 encoding [33]. However, the most efficient way of memorizing a random number is to convert it to a *mnemonic code*. We choose a list of common English words and agree that each word represents a certain sequence of bits. By using this method, a 128 bit number can be represented by 12 English words, which are easy to memorize. Figure 10 shows an example seed and the corresponding mnemonic code in Electrum client.

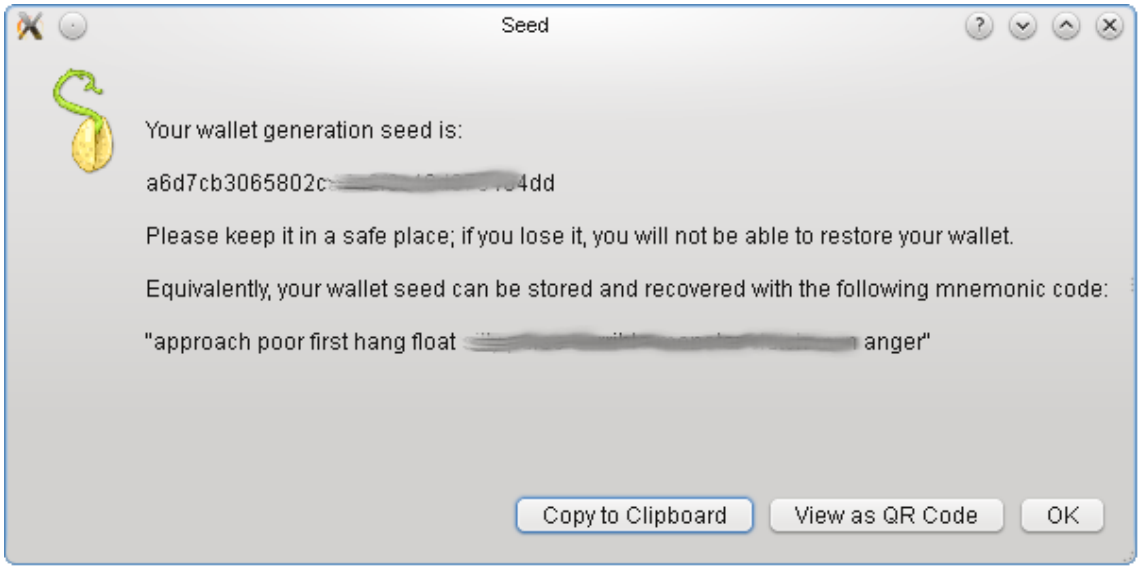


**Figure 10.** Seed and mnemonic code in Electrum

Someone may put some money to a deterministic wallet, remember the seed and remove the original wallet from the hard drive. After these actions have been performed, it is not possible to recover the keys and the money in any way other than to generate the keys from the seed. As the seed does not exist on any physical media, the information on how to access the money is now in the person's mind (brain) and nowhere else. This is why this wallet becomes a *brainwallet*.

## 5.3 Wallet encryption

The threat of malicious software is very prominent nowadays, and any computer can be compromised by viruses, Trojan horses and other types of malware. After Bitcoin became more valuable, cybercriminals created malware that steals the *wallet.dat* file, which holds the private keys for the Satoshi client [34]. Possession of this file enables the attacker to steal all Bitcoins from the addresses in the wallet. If the wallet file is backed up to some location which later becomes accessible by an attacker, it is also possible for him to steal the money.

To mitigate these threats, original Bitcoin client developers introduced wallet encryption in version 0.4.0 [35]. Other clients implemented a similar feature. Users may choose a passphrase, which encrypts the private keys in the wallet, and unencrypted private keys are never written to disk. Figure 11 shows the interface for setting a passphrase in the Satoshi client. If an attacker gains access to an encrypted wallet, it is not possible for them to steal any money from it, assuming the passphrase is not compromised. On

the other hand, it is still possible for malware to steal the money, if the malware runs on the same computer and under the same privileges as the Bitcoin client itself.

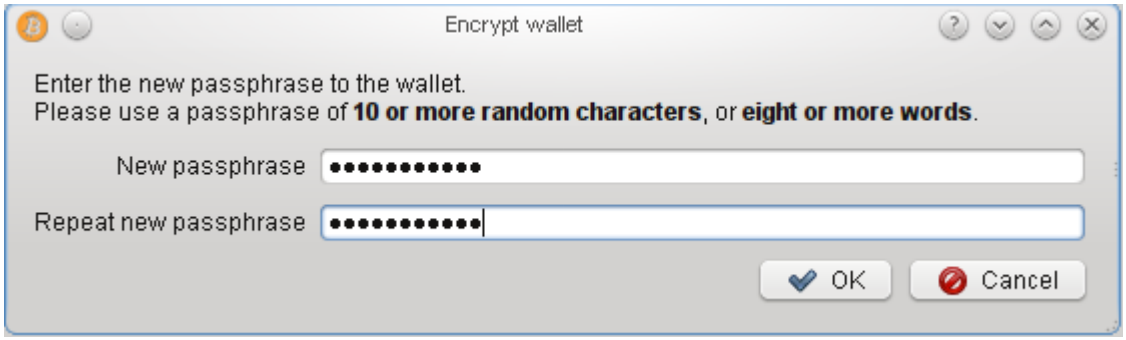


**Figure 11.** Setting a passphrase in the Satoshi client

## 5.4 Watch-only wallets

For merchants, both online and offline, who accept payments from the public, it may be dangerous to keep the wallet with the private keys on a computer that is exposed to the public, such as an Internet server or a computer in a shop. Such computers can become targets of cybercriminals and, if they succeed in infecting these computers with malware, the merchants may have their funds stolen. Publicly exposed computers are more vulnerable to malware infestations than other machines which are not publicly known. Fortunately for merchants, the Bitcoin protocol provides a possibility to accept and track payments without having access to private keys.

The Armory client [36], the Electrum client [31] and some Web-based services such as BitcoinMonitor [41] provide *watch-only wallets*. These wallets have neither private keys nor any information, such as a seed, on how to obtain them. Instead, watch-only wallets have only the public keys and the corresponding Bitcoin addresses. This information is enough to watch the block chain for transactions involving given addresses, but not enough to initiate transactions with them. Consequently, if an attacker gains access to a computer running a watch-only wallet, they will not be able to steal any money. The only useful information an attacker will learn is that certain addresses belong to the same person.

Watch-only wallets can be either randomly-generated or deterministic. The implementation of randomly-generated watch-only wallets is relatively straightforward: the block chain is watched for given addresses. The drawback here is that we need to

either store a large reserve of public keys or send new public keys when they are needed.

The digital signature algorithm used in Bitcoin, the Elliptic Curve Digital Signature Algorithm, provides a possibility to generate an infinite number of public keys deterministically from a *master public key*, which in turn is derived from the seed [42]. Merchants can use the following procedure. First, a seed is generated and stored on a private, secure computer. A master public key is derived from the seed and copied to the public-facing computer. The Bitcoin client on the public-facing computer uses the master public key to generate new addresses when they are needed. After an address is given to a customer for making a payment, the block chain is watched for transactions involving this address. If the public-facing computer is compromised, no money will be stolen because no private keys can be derived from the master public key [42].

## 5.5 Paper backups

There may be reasons to make a backup copy of a Bitcoin wallet not on digital media, but on plain paper instead. Some Bitcoin users prefer to keep their long-term savings on paper to ensure greater security. If the Bitcoin client uses a deterministic wallet, the seed can be printed on paper and put into a safe place. The Armory client [36] is the only one that has paper backup functionality built-in. Figure 12 shows an example paper backup made with the Armory client. The QR code is provided for convenience to avoid having to type letters manually when restoring the wallet from a backup. The Electrum client doesn’t have a built-in printing feature, but its wallet could also be easily backed up on paper by printing the seed manually.

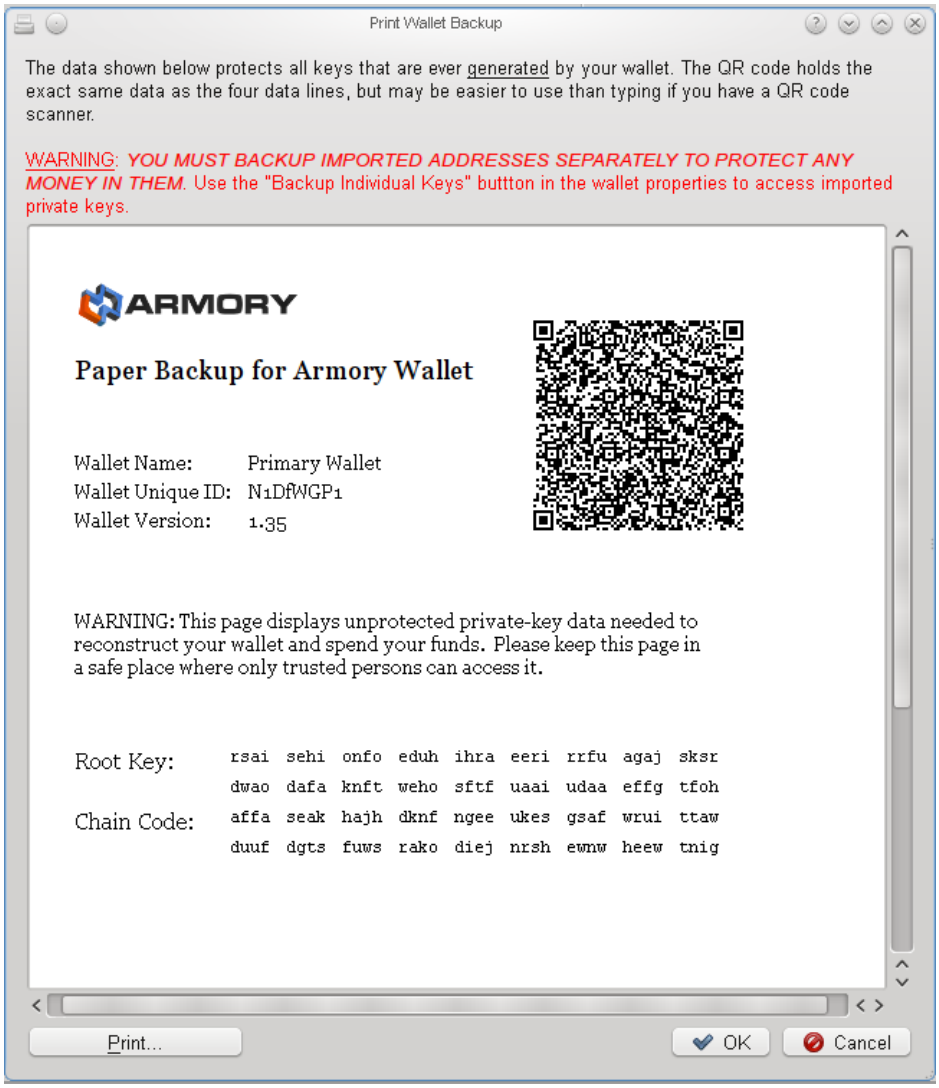


**Figure 12.** Paper backup preview in the Armory Bitcoin client

Backing up wallets with randomly-generated keys is more complicated. Such wallets may have hundreds of keypairs which would occupy multiple pages. Restoring from such a backup would be a long and tedious process. One of the solutions to backing up non-deterministic wallets is to transfer the money to be backed up to one address and print out the corresponding private key. Another approach is to transfer the money to a deterministic wallet and perform the backup as described earlier.

## 5.6 QR codes

There are other uses of papers with QR codes in Bitcoin than backups. Papers with printed QR codes of Bitcoin addresses can be used to accept payments. A poster with a Bitcoin address and some text encouraging donations can be put on a wall in a public

place to raise funds. BitcoinSpinner [21] has the ability to scan QR codes with the phone's built-in camera and, if the scanned code is a Bitcoin address, send money to it.

A paper with a private key printed on it can be used as a means of payment by itself. The person receiving the payment can scan the code, sign a transaction by using the scanned private key and transfer the money to his/her own wallet.

Merchants can show a Bitcoin address as a QR code on a screen to accept payments from customers at the point of sale. In this setup, a watch-only wallet described in Section 5.4 is very useful as it allows confirming the receipt of the money immediately after it is sent while keeping these Bitcoins inaccessible by anyone at the point of sale.

## 5.7 Bitcoin URI scheme

When using a Web browser, we open new pages and download files by clicking on links. Every link on the Web has a URI in it, which tells the browser how to access certain content [46]. Links greatly simplify Web browsing, as entering URIs manually takes much longer time than clicking. To simplify Bitcoin payments and to avoid having to type addresses and amounts manually, the Bitcoin URI scheme was introduced.

As World Wide Web creator Tim Berners-Lee suggested, a URI consists of a *scheme* and a *path*, which are separated by a colon. A path describes the resource itself, and a scheme denotes the namespace for that resource [46]. There are many URI schemes nowadays, and the most popular one is HTTP, which is used to access Web pages. In the recent years, links with the MAGNET URI scheme became a popular way to identify resources in BitTorrent network by their hash [47]. In a similar way, the Bitcoin URI scheme is used to identify addresses and (optional) amounts to be paid. If a merchant requests a customer to pay 1 Bitcoin, the URI may look similar to the following:

```
bitcoin:1LVa9TTgzdNv98JNGWF3v8WsdK2XwmG3io?amount=1X8
```

Several Bitcoin clients, including Electrum [31] and Armory [36] support Bitcoin URIs and fill in the money sending form with the data from the URI when the link is clicked on. Mobile clients, such as BitcoinSpinner [21], can recognize Bitcoin URIs in QR codes and help the user to avoid typing the address and amount manually.

# 6 Future enhancements

## 6.1 Multi-signature transactions

After the introduction of wallet encryption and watch-only wallets, security of Bitcoin clients improved dramatically, but still remains to be a valid concern. During 2011 and the first half of 2012, there have been several large-scale security breaches and heists whose victims were mining pools, Bitcoin exchanges and their customers [48, 49, 50]. The reasons for these unfortunate events were not only negligence and disregard of security practices, but also inherent weaknesses in the current Bitcoin protocol.

These weaknesses stem from the fact that the possession of the private key for a certain address gives ultimate control over all money belonging to that address, and the private key is always needed for signing transactions. To improve security, we may try to implement some multi-factor authentication procedure, as was done by BlockChain.info wallet service: Web-based Bitcoin client requests you to confirm transactions on a mobile phone [22]. Even though this procedure may improve security, in the very end, the benefits of any multi-factor authentication procedure are lost when one of the parties receives the private key to perform signing. If this party is compromised, all security is lost.

The solution to overcome these problems is to use multi-signature transactions. These transactions require multiple signatures to be completed instead of one. We may think of a multi-signature transaction as of a transaction that has more than one recipient address, and several signatures are needed to spend this money further. Gavin Andresen, the main developer of Bitcoin, proposed two new types of transactions: 2-of-2 and 2-of-3 [51]. In a 2-of-2 transaction, money is sent to 2 addresses, and signatures from owners of both addresses are needed to spend this money. In a 2-of-3 transaction, money is sent to 3 addresses, and signatures from any 2 of them are enough to spend this money. The actual implementation is more sophisticated and is described in [52]. To initiate a multi-signature transaction, money is sent to an address which is a hash of 2 or 3 public keys of receiving parties. This address is recorded in the *output* part of the transaction. To spend this transaction later, 2 signatures must be supplied in the *input* part of the next transaction. Figure 13 illustrates the process of creating and spending a 2-of-2 transaction. Owner 1 sends a transaction to the 2nd Owner's multi-signature address, which is generated from the public keys of the 2nd

Owner's home computer and mobile phone. Later Owner 2 spends this transaction to Owner 3 by signing the hash with two private keys: the home computer's private key and the mobile phone's private key.

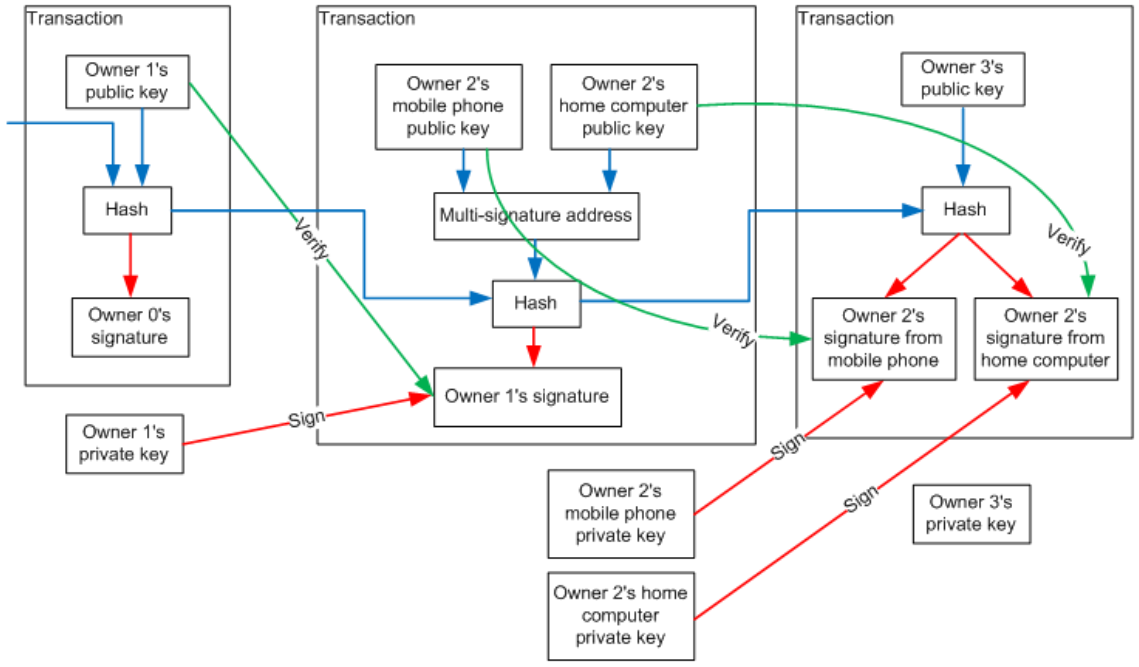


**Figure 13.** 2-of-2 multi-signature transaction creation and spending

The process of constructing multi-signature transactions is more complex than for normal transactions and involves negotiation between several parties. The exact procedures are not defined as of May 2012. In a 2-of-2 transaction, receiving parties need to collaborate to construct the receiving address. One of the possible procedures may include exchanging public keys first, creating the receiving address independently and verifying that the same address was created. When sending this address to the payer, both receiving parties need to be sure that the same address was sent.

The security of Bitcoin wallets can be greatly improved with the use of 2-of-2 transactions. We may store one of the private keys on a computer and another one on a mobile phone. After public keys have been exchanged between the computer and the wallet, we may generate the common receiving address and send money there from a normal wallet. From this moment, if we want to spend the money on the common address, we need signatures from both the computer and the phone. If either the computer or the phone is compromised by an attacker, the money will not be stolen. At the same time, if either the computer or the phone is lost and we do not have the

backup of the data, the money on the common address will be lost. But, fortunately, this drawback can be overcome with 2-of-3 transactions.

A third-party service can be used to hold the third private key in a 2-of-3 transaction. In normal circumstances, we may use the same computer and phone, since 2 signatures are enough to spend the transaction. If one of the devices breaks, we may ask the third-party service to sign our transaction. It should be noted that for better security, any authentication credentials for the third-party service should not be entered on either the computer or the phone, as they may become compromised together with an additional private key. Instead, they should be entered on a completely different device. This procedure ensures both security and availability of Bitcoins in case of a failure.

Another application for 2-of-3 transaction is three-party escrow. If the Buyer purchases certain goods from the Seller, they may choose the Arbiter, trusted by both the Buyer and the Seller to resolve disputes. Before the shipping of goods, a 2-of-3 transaction is initiated. The Buyer sends money to the common address constructed from the public keys of the Buyer, the Seller and the Arbiter. If the purchase runs smoothly and the Buyer is satisfied, he/she gives the signature to the Seller, who adds his/her own signature and gets the money. If the Buyer is not satisfied, the Arbiter has to decide whether the Seller fulfilled the conditions of the purchase. Depending on the Arbiter's decision, he/she gives his/her signature to either the Buyer or the Seller.

Multi-signature transactions also have certain drawbacks: they require sophisticated protocols to be created and occupy a few times more space in the block chain than normal transactions. Deterministic wallets are not usable for multi-signature transactions when third parties are involved where we do not have control over their wallet seeds. In these cases, we need to save copies of public keys involved in those transactions after they happen. This deprives deterministic wallets of their advantages over randomly-generated ones.

At the time of writing (May 2012) multi-signature transactions have only been implemented in the original Bitcoin client, and only in its command-line interface [54]. Even though some proposals for negotiating multi-signature transactions exist [53], a lot of work needs to be done to ensure compatibility and correct operation of the protocol.

## 6.2 Scalability

At the current (May 2012) level of 20 000 transactions in Bitcoin network per day [56], a normal desktop computer is powerful enough to be a full Bitcoin node and to do all activities associated with it. If the popularity of Bitcoin grows further and the number of transactions per day increases, at some point normal home computers will not cope with increased load. A well-known security researcher Dan Kaminsky criticised Bitcoin for lack of scalability [55]. To make Bitcoin more scalable, several optimizations have been proposed.

The author of the BitcoinJ client Mike Hearn suggests using the Simplified Payment Verification procedure, which allows clients to perform verification of transactions without having a full copy of the block chain [19]. Implementing this procedure may significantly decrease the amount of disk space required for full clients to operate. Another way to improve storage efficiency is to remove all transactions that are already spent from the block chain. It has been calculated that doing so reduces the size of the block chain by 71 percent [57].

# 7 Conclusion

Even though Dan Kaminsky characterized Bitcoin as “a really strange use of cryptography” [55], it serves its purpose quite well. Creating an account is as easy as installing a software application. Sending and receiving payments is simple and does not require submitting any documents anywhere. The Bitcoin network is very reliable and as of June 2012 has had only one major disruption since its inception in 2009.

The growth rate of the Bitcoin ecosystem and the underlying technology is tremendous. Bitcoin was first described in 2008 and started operating in 2009. During 2010, GPU mining clients were being developed, and miners gradually switched from CPU mining to GPU mining. In the late 2010, pooled mining was introduced and quickly became the dominant form of mining. After Bitcoin was featured in mass media in 2011, the Bitcoin community and the price of Bitcoins started growing even faster. Unfortunately, the sudden growth of Bitcoin’s popularity led to a series of robberies and data leaks, which severely impacted the public’s trust in Bitcoin and the price of it. One of the reasons for these events was a lack of security features in Bitcoin clients: in the beginning of 2011 wallet encryption and watch-only wallets did not exist. However, a lot of talented software developers joined the Bitcoin community in 2011. As a result, most of the features described in Chapter 5 were introduced, designed and implemented during 2011. Some technologies, such as proof-of-work and deterministic generation of public keys, were used on a large scale for the first time in Bitcoin clients.

Usability and user-friendliness of Bitcoin is improved with deterministic wallets, QR codes and Bitcoin URIs. Wallet encryption, watch-only wallets and paper backups enhance security of Bitcoin clients. Brainwallets boost both usability and security. These features together brought Bitcoin user experience much closer to that of existing payment systems.

Bitcoin is still in its infancy, and its impact on the world’s economics is negligible. We expect both the economics and the technology behind Bitcoin to evolve and expand. Multi-signature transactions, when they are implemented, will provide a vast array of innovative usages of this currency. Since Bitcoin ecosystem develops so fast, it is impossible to predict what will happen to it in the next year. We are looking forward to seeing new ideas and developments.

# REFERENCES

[1] Nakamoto, S. 2008. Bitcoin: A peer-to-peer electronic cash system. Consulted 01.05.2012 http://bitcoin.org/bitcoin.pdf

[2] Bitcoin wiki. Trade. 2012. Consulted 01.05.2012 https://en.bitcoin.it/wiki/Trade

[3] Bitcoin forum. 2010. Consulted 01.05.2012 https://bitcointalk.org/index.php?topic=224.0

[4] Bitcoin charts. 2012. Consulted 01.05.2012 http://bitcoincharts.com/markets/

[5] Chaum, D. 1982. Blind signatures for untraceable payments. Consulted 01.05.2012 http://www.hit.bme.hu/~buttyan/courses/BMEVIHIM219/2009/Chaum.BlindSigForPayment.1982.PDF

[6] Grigg, I. 2005. Triple Entry Accounting. Consulted 01.05.2012 http://iang.org/papers/triple_entry.html

[7] NEXT. 1999. How DigiCash Blew Everything. Consulted 01.05.2012 http://cryptome.org/jya/digicrash.htm

[8] Schneier, B. 1996. Applied Cryptography, Second Edition. USA: John Wiley & Sons, Inc.

[9] Dai, W. 1998. B-money. Consulted 01.05.2012 http://www.weidai.com/bmoney.txt

[10] RowIT. 2012. Bitcoin Peer to Peer Network Status. Consulted 01.05.2012 http://bitcoinstatus.rowit.co.uk/

[11] Back, A. 2002. Hashcash - A Denial of Service Counter-Measure. Consulted 01.05.2012 http://www.hashcash.org/papers/hashcash.pdf

[12] Bitcoin wiki. 2012. Mining. Consulted 01.05.2012 https://en.bitcoin.it/wiki/Mining

[13] BitcoinJ. Source code. 2012. Consulted 01.05.2012 https://code.google.com/p/bitcoinj/source/browse/core/src/main/java/com/google/bitcoin/core/Block.java#584

[14] Standards for Efficient Cryptography. 2000. Recommended Elliptic Curve Domain Parameters. Consulted 01.05.2012 http://www.secg.org/collateral/sec2_final.pdf

[15] Cohen, B. 2008. The BitTorrent Protocol Specification. Consulted 01.06.2012 http://www.bittorrent.org/beps/bep_0003.html

[16] Original Bitcoin client. 2012. Consulted 01.06.2012 https://github.com/bitcoin/bitcoin

[17] Bitcoin version 0.3.22. 2011. Consulted 01.06.2012 http://bitcoin.org/releases/2011/06/05/v0.3.22.html

[18] BitcoinJ. 2012. Consulted 01.06.2012 https://code.google.com/p/bitcoinj/

[19] BitcoinJ. 2012. Security model. Consulted 01.06.2012 https://code.google.com/p/bitcoinj/wiki/SecurityModel

[20] Electrum client. 2012. Consulted 01.06.2012 http://ecdsa.org/electrum/

[21] BitcoinSpinner client. 2012. Consulted 01.06.2012 https://code.google.com/p/bitcoinspinner/

[22] BlockChain.info Wallet. 2012. Consulted 01.06.2012 https://blockchain.info/wallet/

[23] Bitcoin Client API. 2012. Consulted 01.06.2012 https://code.google.com/p/bccapi/

[24] Bitcoin wiki. 2012. MyBitcoin. Consulted 01.06.2012 https://en.bitcoin.it/wiki/MyBitcoin

[25] Instawallet. 2012. Consulted 01.06.2012 https://www.instawallet.org/w/_cmgZpXGT6iE9oh_tX0hhg

[26] MgGox wallet. 2012. Consulted 01.06.2012 https://mtgox.com/

[27] Bitcoin wiki. 2012. Mining hardware comparison. Consulted 04.06.2012 https://en.bitcoin.it/wiki/Mining_hardware_comparison

[28] Bitcoin forum. 2010. Cooperative mining. Consulted 04.06.2012 https://bitcointalk.org/index.php?topic=1976.0

[29] BlockChain.info. 2012. Hashrate distribution. Consulted 04.06.2012 http://blockchain.info/pools

[30] Bitcoin forum. 2012. Key pool feature for safer wallet backup. Consulted 04.06.2012 https://bitcointalk.org/index.php?topic=1414.0

[31] Electrum client. 2012. Source code. Consulted 04.06.2012 https://gitorious.org/electrum/electrum/blobs/master/lib/wallet.py

[32] IETF. 2005. A Universally Unique IDentifier (UUID) URN Namespace. Consulted 04.06.2012 https://www.ietf.org/rfc/rfc4122.txt

[33] IETF. 2006. The Base16, Base32, and Base64 Data Encodings Consulted 04.06.2012 https://tools.ietf.org/html/rfc4648

[34] F-secure. 2011. Pickpocket Targets Wallets at Bitcoin Forum. Consulted 04.06.2012 https://www.f-secure.com/weblog/archives/00002187.html

[35] Bitcoin version 0.4.0. 2011. Consulted 04.06.2012 http://bitcoin.org/releases/2011/09/23/v0.4.0.html

[36] Armory Bitcoin client. 2012. Consulted 04.06.2012 http://bitcoinarmory.com/

[37] Libbitcoin. 2012. Consulted 04.06.2012 https://gitorious.org/libbitcoin/libbitcoin/

[38] Multibit Bitcoin client. 2012. Consulted 04.06.2012 http://multibit.org/

[39] Bitcoin forum. 2011. Phoenix miner. Consulted 04.06.2012 https://bitcointalk.org/index.php?topic=6458.0

[40] Bitcoin forum. 2011. CGMiner. Consulted 04.06.2012 https://bitcointalk.org/index.php?topic=28402.0

[41] BitcoinMonitor. 2012. Consulted 05.06.2012 http://www.bitcoinmonitor.net/

[42] Bitcoin forum. 2011. Deterministic wallets. Consulted 05.06.2012 https://bitcointalk.org/index.php?topic=19137.0

[43] Shen, X.; Yu, H.; Buford, J.; Akon, M. 2010. Handbook of peer-to-peer networking. USA: Springer Science + Business Media

[44] Bitcoin wiki. 2012. Block chain. Consulted 05.06.2012 https://en.bitcoin.it/wiki/Block_chain

[45] Bitcoin wiki. 2012. URI scheme. Consulted 05.06.2012 https://en.bitcoin.it/wiki/URI_Scheme

[46] Berners-Lee, T. 1994. Universal Resource Identifiers in WWW. IETF. Consulted 06.06.2012 https://tools.ietf.org/html/rfc1630

[47] Hazel, G.; Norberg, A. 2008. Extension for Peers to Send Metadata Files. Consulted 06.06.2012 http://bittorrent.org/beps/bep_0009.html

[48] Bitcoin forum. 2011. “bitcoin7.com 'hacked'. Database and wallets 'stolen'” Consulted 06.06.2012 https://bitcointalk.org/index.php?topic=46982.0

[49] Bitcoinmedia. 2012. Compromised Linode & coins stolen from slush, faucet and others. Consulted 06.06.2012 http://bitcoinmedia.com/compromised-linode-coins-stolen-from-slush-faucet-and-others/

[50] The Bitcoin Trader. 2012. Developing: Bitcoinica "Hacked" - Potentially 18,000 BTC ($90,000 USD) Stolen. Consulted 06.06.2012 http://www.thebitcointrader.com/2012/05/developing-bitcoinica-hacked.html

[51] Andresen, G. 2011. M-of-N Standard Transactions. Consulted 06.06.2012 https://en.bitcoin.it/wiki/BIP_0011

[52] Andresen, G. 2012. Pay to script hash. Consulted 06.06.2012 https://en.bitcoin.it/wiki/BIP_0016

[53] Reiner, A. 2011. Multi-Sig Transaction Distribution. Consulted 06.06.2012 https://en.bitcoin.it/wiki/BIP_0010

[54] Bitcoin version 0.6.1. 2012. Consulted 06.06.2012 http://sourceforge.net/projects/bitcoin/files/Bitcoin/bitcoin-0.6.0/

[55] Kaminsky, D. 2011. Some thoughts on Bitcoin. Consulted 06.06.2012 http://www.slideshare.net/dakami/bitcoin-8776098

[56] Blockchain.info. 2012. Number of transactions per day. Consulted 06.06.2012 http://blockchain.info/charts/n-transactions

[57] Bitcoin forum. 2011. Script calculates 71% freeable transactions. Consulted 06.06.2012 https://bitcointalk.org/index.php?topic=9461