UMEÅ UNIVERSITY
Department of Computing Science
Master thesis report

Master degree project in computer science
30HP, 5DV143, VT19

Cross-chain Settlement and Exchange in Cryptocurrency

Name	Carl-Johan Andersson
CAS	caan0156@umu.se
CS	c14can@cs.umu.se
Date	2019-03-06

Abstract
This is empty for now

University Supervisor
Jan-Erik Moström (`jem@cs.umu.se`)
Company Supervisor
Oskar Janssssskebab son (`oskar.janson@cinnober.com`)

Glossary

Crypto currency - A currency backed not by centralized authorities but by mathematical evidence and clever mechanisms
Bitcoin - The very first and most mature cryptocurrency on earth
satoshi - The smallest fraction of a bitcoin. 100.000.000 satoshis = 1₿
Satoshi Nakamoto - The pseudonym used by the creator of bitcoin
Proof-of-work - A system where a someone can prove mathematically that work was put into doing something.
Mining - In this context refers to looking for a valid hash in a proof-of-work system. Most often by checking random numbers in the nonce field until a valid hash is found
Atomic (Adjective) - Something that only has two outcomes. Either completed fully or no changes to the state at all. An atomic task can not be half completed

More to come...

Contents

Chapter 1

Introduction

Although blockchain and other types of distributed ledgers are still in their infancy a growing number of people and companies are starting to see that they hold great promise, especially when it comes to financial technology, where things like trust and security is held to be very important.

1.1 The origins of Bitcoin

The most well known, developed and researched blockchain technology is Bitcoin. The mysterious nature of Bitcoins creator makes it hard to pinpoint how and when the idea was first thought up and when the development started, the most exact way would be to pinpoint at: `2009-01-03 18:15:05`[1], which is the timestamp in the very first block in the Bitcoin blockchain, however the white paper (**Bitcoin: A Peer-to-Peer Electronic Cash System**)[11] specifying the technical details circulated on cryptographic mailing lists as early as 31 October 2008, and the domain name `bitcoin.org` was registered 18 August 2008.[8]

1.1.1 Satoshi Nakamoto

The author name given in the white paper is `Satoshi Nakamoto`, this name is believed to be a pseudonym. Satoshi remained in the bitcoin community for a couple of years, regularly posting on the forum `bitcointalk.org` and keeping up with conversations in the mailing list. Those who have been interested in finding out Satoshi's real identity have analyzed his active time and language used. The findings were that Satoshi was most active during Western European day time, and he also used a lot of Anglo-colloquialisms such as "bloody hard" [4], and "flat" instead of apartment, so a popular theory is that Satoshi lived in Britain at least during this time.[8]

`April 23, 2011` was the last time anyone ever heard from Satoshi Nakamoto, in a mail to a fellow developer Mike Hearn he said "I've moved on to other things. It's in good hands with Gavin and everyone.".[6] Speculations on who Satoshi really is are still going strong even to this day, but Nakamoto's true identity is so far unknown.[8][9]

1.2 Basics on Bitcoin

Most people, even the layman with no blockchain experience, have at least heard of Bitcoin. But the exact details of how it works is not common knowledge. Described in a single sentence: Bitcoin is a currency where a communal ledger, that is shared between all participants in the whole world, dictates who owns what in terms of money or other assets.

The regular monetary system we are used to has a centralized authority, for example a central bank, who decides how much money is in circulation, who can transact with who etc. The monetary system presented by Bitcoin has no centralized authority, instead it relies on decentralized, trust-less verification.

These usually are the main concerns people have with distributed ledgers:

- How is spending someone else's money prevented?
- What prevents someone from "printing" more Bitcoins?
- How is spending the same money in different places in the world prevented?
- How is the consensus on the order of transactions reached?
- How do I interact with it?
- What type of transactions can you make?

These questions will be given ansers in the sections below.

1.2.1 Digital signatures

Bitcoin would not be possible at all without the underlying cryptographic mathematics. When it comes to proving ownership in Bitcoin ECDSA (Elliptic curve digital signature algorithm)[12] is used. Elliptic curve cryptography will not be covered in depth here but the basic idea is that you have a private and public key. The public key can be shared with anyone without danger, the public key can be derived from the private key, but not the other way around, this is something called trap-door mathematics.[12][7] This means that it is easy to go one way via equations. But going back is nearly impossible.

The reason for it not being completely impossible is because any potential attacker could always just keep guessing private keys until the right one is found. However with a sufficiently large private key, let's say 256-bits, and a computer that could check one billion billion (10^{18}) private keys every second (If such a machine could exist at all) it would still take $\frac{(2^{256}/10^{18})}{(60 \cdot 60 \cdot 24 \cdot 365)} \approx 3.67 \cdot 10^{51}$ years to check all possible private keys.

The most common transaction made on the blockchain is one where you "pay to" a public key. Who ever holds the private key paired with that public key can then via a mathematical equation prove that they hold the private key without actually revealing the private key.[5]

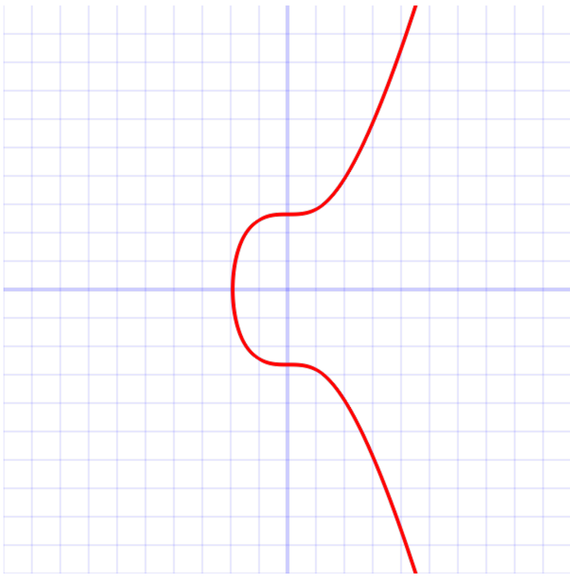


Figure 1.1: The `Secp256k1` plotted over real numbers. Note that the real curve is over a field, and thus looks more like a scattering of random points

The elliptic curve used can hold different parameters that defines it, certain elliptic curves are standardized and have their own names. The curve used in Bitcoin is named `Secp256k1`.[13][7]

The typical curve used in Elliptic curve cryptography is on the form $y^2 = x^3 + ax + b$. The `Secp256k1` is defined with $a = 0$ and $b = 7$, making the full `Secp256k1` equation: $y^2 = x^3+7$. Which is plotted in figure 1.1. For more in depth on elliptic curve cryptography see section 2.3

1.2.2 Blocks and the blockchain

A block is fairly simple to understand, it is simply a datatype or structure that holds information about itself, all the transactions that can fit and the previous block in the chain (more on that in a bit). Because every block holds information about the previous block you can follow all blocks backwards in time all the way back to the original, also called the genesis block.[1] This is what is referred to as the blockchain.

A block could be added to the chain by anyone in the world. However it will only be accepted if it has sufficient proof of work, and this is the key to how consensus is reached in the network.[7]

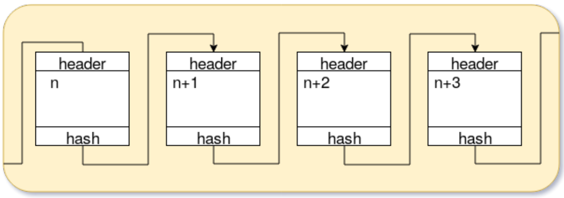


Figure 1.2: A basic overview of a blockchain

Block-header

Each block has a section of data called a header. The header contains meta-data about the block itself such as the version number, the id of the previous block in the chain, a timestamp of when the block was mined, the merkle root of all transactions (serves as proof of what transactions was included in the block) and a 32-bit field called nonce.

The size of the header is always 80-bytes, a blocks id[1] is equal to the hash of its header. For example the genesis block in the Bitcoin blockchain has the id:

```
000000000019d6689c085ae165831e934ff763ae46a2a6c172b3f1b60a8ce26f
```

Something that you will find with all block-ids is that they always have leading zeroes. This is a side-effect caused by mining and proof-of-work as explained in section 1.2.3.

1.2.3 Proof of work

Proof of work is, just like the digital signatures, based in cryptographic mathematics. Before we go on you first need to understand what a hashing algorithm does, the hashing algorithm used in Bitcoin is called `SHA256`. `SHA256` takes in data of any size and produces a sort of finger print of 256 bits.

For example the `SHA256` of the text "cool":

```
echo "cool" | openssl sha256
```

[1]Block id and block hash refers to the same thing. The terms might be mixed throughout the report.

Produces:

```
27c16ce7e3861da034af1bb356d6a4f38cb84fa65d51fa62f69727143b4c6b60
```

The text produced is actually bytes represented in a hexadecimal number system, in fact the entire string can be considered to be a very large number. Just like with the digital signature there is no known viable way that can take a hash and find what the original data that produced it was.

There is a term in Bitcoin called mining difficulty, or just difficulty. This is a large number, 256 bits to be exact. When you want to add a new block to the chain you have to do something called mining, this is a process where you change the nonce-bits in the block-header until the id of the block (Think of this a number) is less than the target difficulty. The term hashpower refers to how many times the machine you are using to mine blocks can test a certain combination of variables per second, or H/s (Hashes per second).

The difficulty of mining a block is adjusted about every 2 weeks. The difficulty is set so that the combined hashpower of the entire world is enough to mine one block every 10 minutes on average.[7]

The accepted order of transactions is the order going backwards from the latest block on the longest chain. The longest chain is always the one that the majority is mining towards. This is simplified to the extreme but what it basically means is that as long as you trust 51% of the participants in the network you can also trust that the order of transactions is correct. This mechanism prevents things like double spending the same money and holds a property called emergent consensus, which means that eventually the entire network will agree on the order of transactions.

In figure 1.3 is a diagram showing the longest chain, meaning the chain with most proof of work. The block in green is the latest block on that chain. The yellow block is what is known as a stale block, a block that is part of the chain but not part of the longest chain of blocks. Transactions in a stale block are not considered valid, and eventually they can be pruned (deleted) because they do not effect the future in any way. A stale block occurs if a new block is found in different parts of the network at almost the same time. Then the network will be split, each node tries to find the next block on the chain of whatever block they received first. In the case shown below the chain on the left "won" and the green block is now the longest chain.

The red block in figure 1.3 is what is known as an orphan block, that is it has no known parent in the chain. This can happen if someone mines a block that is malformed or when building the chain for a new node and the blocks are received out of order.

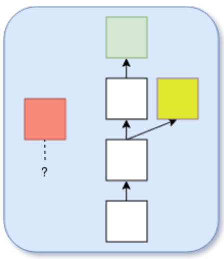


Figure 1.3: A diagram showing the longest chain, The green block is the latest block on the longest chain. The red block is a an orphan block, the yellow block is a stale block

1.2.4 Software

Just like how you can use regular money knowing the underlying process and technology of for example the banking system, you can use Bitcoin without knowing the details of how it works. There is plenty of software that handles wallets, transactions etc for you.

A good example is mycelium wallet for Android.

1.2.5 Programmable transactions

Another feature that Bitcoin has, which makes it very versatile, is that transactions are programmable As mentioned earlier the most common transaction is sending the money to someones public key. What really goes on here is that the person who wants to spend whatever money was sent to them has to prove programmaticlly that they own the private key related to that public key.

This is not the only type of transaction possible, Bitcoin has its own little programming language called `Script`. Any type of transaction that can be described in script is possible as a transaction. This is basis for atomic swaps, payment channels and lightning network, which will be discussed in the following sections

1.3 Payment channels

One of the biggest problems facing Bitcoin is scalability. At time of writing onchain transactions is capped at about ≈ 7 T/s (transactions per second).[2] The throughput is limited by network propagation and a hard cap on block sizes., a block can only contain so many transactions before it is full. There are a couple of proposed solutions to this however, and one of them is connected payment channels

First off, a payment channel in Bitcoin is a type off trick using programmable transactions, were two users can open a bidirectional payment channel where an infinite number of transactions could be trustlessly exchanged without using the blockchain.

A channel can be opend by one or both participants by using a funding transaction. The funding transaction requires the signature of both participants to spend and is transmitted to the blockchain. After that the participants in the channel exchange commitment transactions that represent exchange in money. Any of the commitment transactions could be broadcast to the blockchain whenever a participant want to close the channel. Once a channel has been closed it cannot be used for further transactions. Clever mechanisms exists in the creation of these channels that makes it so one participant can't cheat the system and spent moeny that does not belong to them.

Just payment channels alone were not enough to fix the scaling problem however, as a payment channel only allows two parties to exchange unlimited transactions. A proposed extension to the payment channels is the lightning network.

1.3.1 Lightning network

Lightning network is a relatively recent development in the Bitcoin community. Payment channels has been known about for a while. But in January of 2016 a white paper was released detailing a promising new extension.[10] It showed that with a few changes to the Bitcoin protocol a new type of payment channel could be opened that allows transactions to propagate through multiple channels.[10]

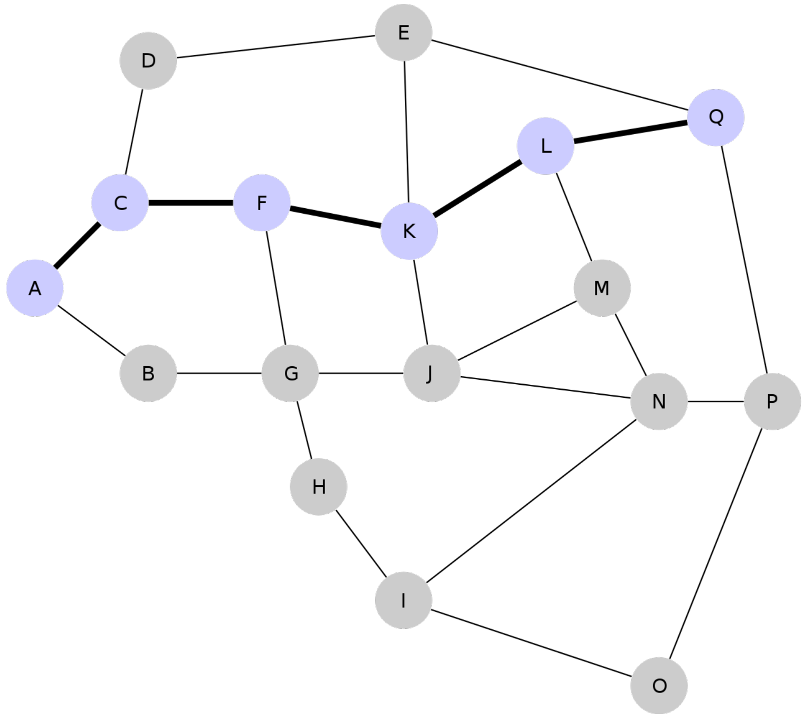


Figure 1.4: A basic overview of a Lightning network, each node represent someone and each edge represents a channel between two people.

Lightning network is really just a network of peers connected via payment channels. In figure 1.4 is an example. Let's say that Alice (node A) want to send a transaction to Quentin (node Q) but they have no direct payment channel between them. With lightning network they can send the transaction via the peers that are between them.

1.4 Atomic swaps

Another recent development in bitcoin and other cryptocurrency is atomic swaps. A regular swap can for example be two parties exchanging currencies. Before this could be done in person, or via a trusted third-party handling all the transactions. Atomic swaps however is as the name implies atomic. Meaning that they either go through completely or no assets change hands at all, they are also completely trust-less meaning that you don't have to trust neither the other party or any third-party.

Just like lightning network atomic swaps use clever transaction scripts to achieve new functionality. Atomic swaps have been shown to be possible over lightning network, but it is mostly small projects and random writings on forums etc

Atomic swaps was first described by Tier Nolan in a post on `bitcointalk.org` on May 21, 2013.[3]

1.5 The goal of the project

The goal of the project is to evaluate all the methods of atomic swaps available and compare them to a couple of different scenarios and use cases. A secondary goal is to master payment channels, atomic swaps and the lightning network

Chapter 2

Bitcoin and Smart contracts

This section is far from finished.

2.1 Bitcoin: a peer to peer electronic cash

As described in the title of the original white paper bitcoin is a peer to peer electronic cash system, which does not rely on any centralized third-party to neither verify the validity of any transactions or handling of transaction completion. Instead it is entirely decentralized and trust-less. The mechanisms and mathematics that makes this possible are relatively recent discoveries in computer-science.

2.1.1 Network wide consensus

One of the main problems facing decentralized currencies before bitcoin was thought up is the byzantine generals problem (Byzantine fault). This refers to independent agents in a system being unable to reach a consensus on what has transpired and what actions to take next. The problem can be imagined as the network being split, where one subsection of the network believes transaction order **A** is correct while another non-overlapping subsection thinks transaction order **B** is correct. How can this be resolved, and how will a new node joining the network know which order is the right one.

Bitcoin was the first cryptocurrency to properly solve this problem once and for all, with the help of something called proof-of-work. proof of work originates from the slightly older idea of hashcash.

Hashcash was/is a means to limit email spam and DDoS-attacks by a proof of work system. For example a sender could be required to produce a hash of message and nonce with a certain number of bits set to 0 at the start of the hash sequence. This (statistically) should take several attempts to produce. But correctness could be checked in a single step. Thus the hash sent together with the message could be considered proof of work, because there is no known way to produce such a valid hash of a message without trying it randomly, thus the only realistic way to have such a hash is if you worked for it. Looking for a valid hash in this kind of proof of work systems is often referred to as **mining**.

Bitcoin used this proof of work concept for another purpose however. Rather than combating spam the proof of work mechanism is used for reaching consensus. When ever a new block is added to the chain it needs to meet a certain proof of work requirement, called difficulty. This difficulty is set so that it should take the combined hashing power of all participants on average 10 minutes to find a valid hash of the next block in the chain.

There is a term called longest proof of work in bitcoin, this refers to the chain of blocks that has the most hash-power supporting it. The chain with the most work done is statistically the one with the most participants. As long as 51% or more of the participants in the network are honest the longest chain can be trusted. So any new members can accept the chain with most work as the truth.

Splits

Contention for the longest chain can arise if a new block is found in two different parts of the network at (almost) the same time. This is not a problem and will eventually be resolved. If you imagine two blocks (**A1** and **B1**) being mined in different parts of the network with the same parent block and half the network got **A1** first and the other half of the network got **B1** first. While the entire network will accept all valid blocks they will only mine towards continuing the chain on the block they received first. So if a new block **A2** with the parent block **A1** is found first the chain formed by block **B1** will be considered invalid and the network continues the chain on the **A** side.

Such contention is called a split, or a fork, and happens naturally once every week or so. As explained they will eventually resolve themselves. The split becomes increasingly more unlikely to survive the longer it goes on. To begin with it is unlikely that a new block will form a split in the first place, and for the split to survive another block both new chains will have to receive a new block at almost the same time.

In common bitcoin lingo there is no difference between a split and a fork. But in this report a **split will referred to as one occurring unintentionally** and **a fork being intentional** just so there is no confusion

> **How long was the longest split?**
>
> The longest splits that occurred by chance were 4 block long and has occurred at least at 3 different occasions.
>
> The longest split ever was caused by an update to the bitcoin core reference implementation (**0.8.0**) that rejected a block that the other implementations did not reject, the nodes accepting the new block keept building on it while those who had updated built on a different chain. The split lasted for 52 blocks before it was resolved.

Forks

Forks happen when there is a disagreement in the bitcoin community when it comes to protocol and consensus. The most famous fork in all of cryptocurrency occurred on `1st August 2017` and was caused by a conflict regarding the size of blocks. How bitcoin will scale to world-wide use has always been a hot debate in the bitcoin community, the majority seeks to scale bitcoin via second layer solutions such as payment channels, lightning network and side-chains. However there were those who disagreed, and instead wanted the network to handle more transactions by making the blocks larger at cost of centralization.

A change in the blocksize requires the entire network to upgrade to the new protocol rules, but the block size increase was not accepted by enough nodes. So the group decided that a fork was the only way to resolve the issue. So as mentioned on `1st August 2017` the first block was mined on the new chain.

The new chain got the name **Bitcoin cash** (bcash) while the main chain still is called just **Bitcoin**. Most miners and node owners stayed with Bitcoin, but a fraction jumped ship and started working on bcash instead. Today both chains runs along side each other, co-existing.

Forks are considered a valid way to vote on what consensus rules and protocol changes should be made. When a change is planned to be made in the chain those who disagree can branch off and follow the old rules. Thus in a way everyone gets their way. You could even branch off on your own forked-chain alone. The main problem for those forking is that most vendors and users still consider the largest (most users, most developers, largest price, etc...) chain to be the valid one.

Soft and hard-forks

TODO

2.3 Elliptic-curve cryptography & ECDSA

As covered in the introduction, Elliptic-curve cryptography (**ECC**) and ECDSA is a fundamental building block of bitcoin. Elliptic curve cryptography relies on intractability of calculating the discrete logarithm of a elliptic curve element with respect to a publicly known base point. Or put another way: It is easy to calculate elliptic curve multiplication with multiplicand n. But calculating n from the resulting point is considered infeasible with sufficiently large curves and multiplicands.

An elliptic curve is defined by the equation $Y^2 = x^3 + ax + b$ and six domain parameters $E(p, a, b, G, n, h)$. **p** is the field that the curve is defined over, this is usually a very large prime number. The curve being defined over a field simply means that the points on the curve fall within $[0, p]$ rather than within the real numbers $\mathbb{R}$. In other words the curve is defined over the field $\mathbb{F}_p$. **a** and **b** are whatever number you put into the equation. **G** is the generator point, that is the point on the curve that will be used in point multiplication later. **n** is the order of G. What that means is that n is the largest number that G can be multiplied by before a point at infinity is produced. n pretty much tells you the limit on how points on the curve that can be generated from G. **h** is the co-factor of the curve. It can be calculated as follows: $h = \frac{1}{n}|(E(\mathbb{F}_p)|$, where $|(E(\mathbb{F}_p)|$ is the order/cardinality of the group of points possible on the curve over field $\mathbb{F}_p$. n is derived from G, G and p should be chosen in such a way that $h \leq 4$, preferably $h = 1$.

These domain parameters can be chosen manually or you can use predefined parameters. Elliptic curves that used predefined domain parameters are called named-curves. The named curve used by Bitcoin is called `Secp256k1`

2.3.1 Secp256k1

`Secp256k1` is defined with the following domain parameters (hexadecimal):

$p =$ `FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFE FFFFFC2F`
or alternatively:
$p = 2^{256} - 2^{32} - 2^9 - 2^8 - 2^7 - 2^6 - 2^4 - 1$

$a = 0$
$b = 7$

$G =$ (`79BE667E F9DCBBAC 55A06295 CE870B07 029BFCDB 2DCE28D9 59F2815B 16F81798`,
`483ADA77 26A3C465 5DA4FBFC 0E1108A8 FD17B448 A6855419 9C47D08F FB10D4B8`)

$n =$ `FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFE BAAEDCE6 AF48A03B BFD25E8C D0364141`
$h =$ `1`

2.3.2 Math on the elliptic curve

Two mathematical operations needs to be defined to operate on the elliptic curve: addition and multiplication

Point addition
Let's say you have to distinct points P and Q that both fall on curve $E(p, a, b, G, n, h)$ $(Y^2 = x^3 + ax + b)$.

$$P + Q = R \Rightarrow (X_P, Y_P) + (X_Q, Y_Q) = (X_R, Y_R)$$

$$X_R = \lambda^2 - X_P - X_Q$$
$$Y_R = \lambda(x_P - X_R) - Y_P$$

where λ:

$$\lambda = \frac{Y_Q - Y_P}{X_Q - X_P} \mod p$$

Point multiplication
If P and Q are coincident, meaning that they have the same coordinates the equation is slightly different.

$$P + Q = R \Rightarrow P + P = R \Rightarrow 2P = R$$

This could be seen as P being multiplied with scalar 2. Most of the equation is the same as with addition, the difference is that:

$$\lambda = \frac{(3X_P^2 + a)}{(2Y_P)} \mod p$$

Faster multiplication with large scalars
Take $xP = R$ that could be calculated by summing P x times:

$$\sum_{n=1}^{x} P = R$$

This might work fine for smaller numbers but for a very large number, like $x = 2^{100}$ it will take infeasible amount of time to calculate. Luckily there is a convenient short cut that you can take called double and add.

First remember that: $P + P = 2P \Rightarrow 2P + P = 3P \Rightarrow 4P = 2(2P) \Rightarrow 8P = 2(2(2P))$

Lets say $x = 200$ in binary terms this could be written as $x = 128+64+8$ or $x = 2^7+2^6+2^3$ thus $200P = R$ could be written as

$$2^7P + 2^6P + 2^3P = R$$

which could be shorten to:

$$2(2(2(2(2(2(2P)))))) + 2(2(2(2(2(2P))))) + 2(2(2P))$$

which looks cumbersome but now instead of 200 calculations you only have to do 18.

2.3.3 Private and public key

Just as RSA cryptography, ECC relies on public-private key encryption and signatures. The public key can be shared freely to everyone, while the private key should, as the name implies, be kept private. Each unique private key has a corresponding public key, through mathematics it can be proven that someone holds the private key paired with a certain public key, without actually revealing the private key.

In ECC **a private key is a really large number**. Imagine you have curve $E(p, a, b, G, n, h)$ and you want to generate a brand new private key k. k could be any number between 0 and n. Any $k > n$ will produce the exact same public key so that will not work. **A public key in ECC is represented by a point in 2D space**, more specifically a point that falls on the curve. To generate a public key P from a private key k you perform $kG = P$ as described in the section above.

Compressed key

The public key is quite large, with two 256-bit numbers representing coordinates. But there is a clever trick we can use to compress the size of the key. Take the Secp256k1 curve for example ($Y^2 = x^3 + ax + b$). It is mirrored around the x-axis, meaning that for each x value there are two possible y values. Thus a public key can be represented by only it's x value plus a prefix telling you which resulting y-value to choose.

Note that because y and x is over $\mathbb{F}_p$ there is no negative value, instead the y value is referred to as even or odd.

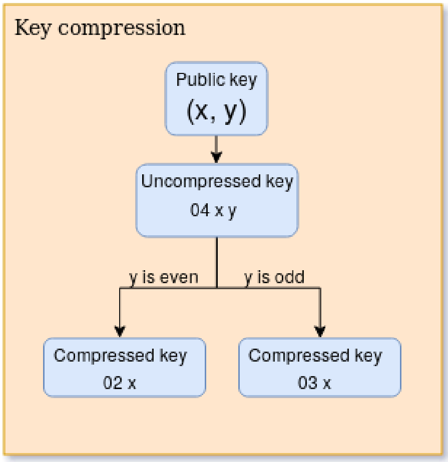


Figure 2.1: How to compress the public key in ecc

2.3.4 ECDSA

The main usage of ECC in cryptocurrency is for proving ownership of coins. The proof relies on elliptic curve mathematics like before. Lets say **Alice** has a message m and want to send it to **Bob** and also prove that the message came from her. First of let's establish some variables: k_A is the private key belonging to Alice, from that private key P_A was generated (The public key), that Bob knows about.

Signing

First calculate the hash of the message:

$$e = HASH(m)$$

If e has a bit-length (numbers in binary representation) that is longer than the bit-length of order n of the curve used. e has to be trimmed down so that the bit-lenghts match

Select a cryptographically-secure random number z that falls in the range $[1, n-1]$ and calculate a new curve point: $(x_1, y_1) = z \times G$

Calculate r and s such that: $r = (x_1 \mod n)$ and $s = (z^{-1}(e + r \times k_A) \mod n)$. if either r or s ends up being 0, generate a new z and try again.

The signature will be the point $(r, s) = S_A$.

Signature validation

If **Bob** wants to verify that it was actually **Alice** that signed the message m. He first has to do a sanity check on the signature S_A to make sure that it is a valid point on the curve and that s and r is within the range $[1, n-1]$ etc...

Calculate the hash e of m the same way as it was done during the signing process. Calculate

$$w = (s^{-1} \mod n)$$

and

$$u_1 = (ew \mod n)$$
$$u_2 = (rw \mod n)$$

From u_1 and u_2 calculate the point $(x_1, y_1) = u_1 \times G + u_2 \times S_A$

The signature is valid if and only if $r = x_1 \mod n$

2.4 Script

Script is the name of the programming language used in Bitcoin and its derivatives. It was not used to write the Bitcoin implementation but rather it is what makes transactions in Bitcoin so versatile. This section however will not focus on where or how Bitcoin uses Script, but rather on how Script itself functions.

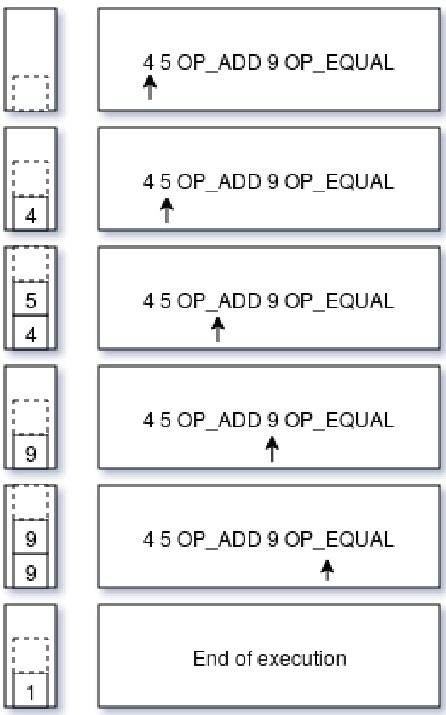


Figure 2.2: Execution of simple program, to the left is the stack, to the right is the program with execution pointer

Script is a forth-like stack based language that is not turing-complete. Turing-completeness means that a language can do anything that the imaginary turing-machine could do, in other words it basically means is that the language can do any mathematically sound operation. Script is **NOT** turing-complete on purpose, more on why can be found in the section about transactions (2.5). A good example is loops, in most languages there is some sort of structure that allows for a piece of code being executed repeatedly. In Scipt this is strictly disallowed, as it has neither for-loops or while-loops.

As mentioned earlier. Script is a stack-based language. That means that as the language executes it uses a stack to store data and variables. Do not confuse the term with the heap and stack from regular programming language discourse. In script there is no heap, instead the stack is the only form of memory, and it acts just as you would expect from a stack, to add a value you have to push it to the stack and to read a value you have to pop it from the stack.

The language it self is quite basic. It relies on operation codes (op codes) the size of a single byte. Most operations pop values from the stack, does something with the values, then pushes the result back on the stack. When Script is written out on paper many op codes and values are excluded, because they are implicit. For example:

```
OP_PUSHDATA1 4 FFFFFFFF
```

This operation pushes 4 bytes to the stack, the bytes all have the hexadecimal value `FFFFFFFF`. But usually when this operation is written out it is shorten to just:

```
FFFFFFFF
```

Whenever a hexadecimal value appears in the code it is implicit that that value is pushed to the stack in the form of bytes. Here is annother program:

```
4 5 OP_ADD 9 OP_EQUAL
```

This simple program will push 4 and 5 to the stack, `OP_ADD` pops two values from the stack (4 and 5) adds them together and pushes the result back on the stack. Then a 9 is pushed to the stack. `OP_EQUAL` pops two values form the stack and pushes a 1 if they were equal, 0 otherwise. In this case a 1 will be left on the stack as $4 + 5 = 9$. Figure 2.2 visualizes what happens at each step of execution.

2.4.1 Complex operations

There are some operations in Script with slightly higher complexity, which do not act like the others. One of them is `OP_VERIFY`, this will perform a verify check on the script. It will pop one value from the stack, and if that value equates to false execution will end immidietly and the entire script will be marked as invalid (see section 2.4.2). If the value equals true execution will continue where it left off. Some operations are combinations with the verify check, like for example `OP_EQUALVERIFY`. This is equal to writing `OP_EQUAL OP_VERIFY`, meaning that it first does an equal check then verifys the result.

There are several operations for checking signatures. These are not so complex in terms of what they do in the script. Their implementation is quite complex however. They reliy on outside information that is not present in the script to check the validity of a signature. The two most used are `OP_CHECKSIG` and `OP_CHECKMULTISIG`. These will not be covered in full in this section as they are more related to transactions so see section 2.5 for a full explanation.

2.4.2 Valid and invalid scripts

At the end of execution a script is marked as either valid or invalid. A script is invalid for the following reasons:

- The stack is empty
- There are more than one values on the stack
- The only value on the stack equates to false
- A VERIFY check fails sometime during execution.

A script is valid if and only if there is one value on the stack, and it is not equal to false.

2.4.3 Bug in Script

An interesting bit of trivia is that there is a bug in the language implementation. More specifically with the operation `OP_CHECKMULTISIG`. The bug makes it so this operation pops one more value from the stack than it is supposed to. This was not discovered until the network had been running for a while. It can't be easily fixed as it is now a part of the consensus rules. All implementations of Bitcoin node has to implement the bugged version of this op code otherwise consensus on the validity of transactions will break.

A fix to this bug would require a hardfok, see section 2.1.1, and has so far been considered not worth it.

2.5 Transactions

Transactions in Bitcoin are not as straight forward as you might expect a transaction to be. A transaction contains a list of inputs and a list of outputs as well as some metadata like version number and lock-time.

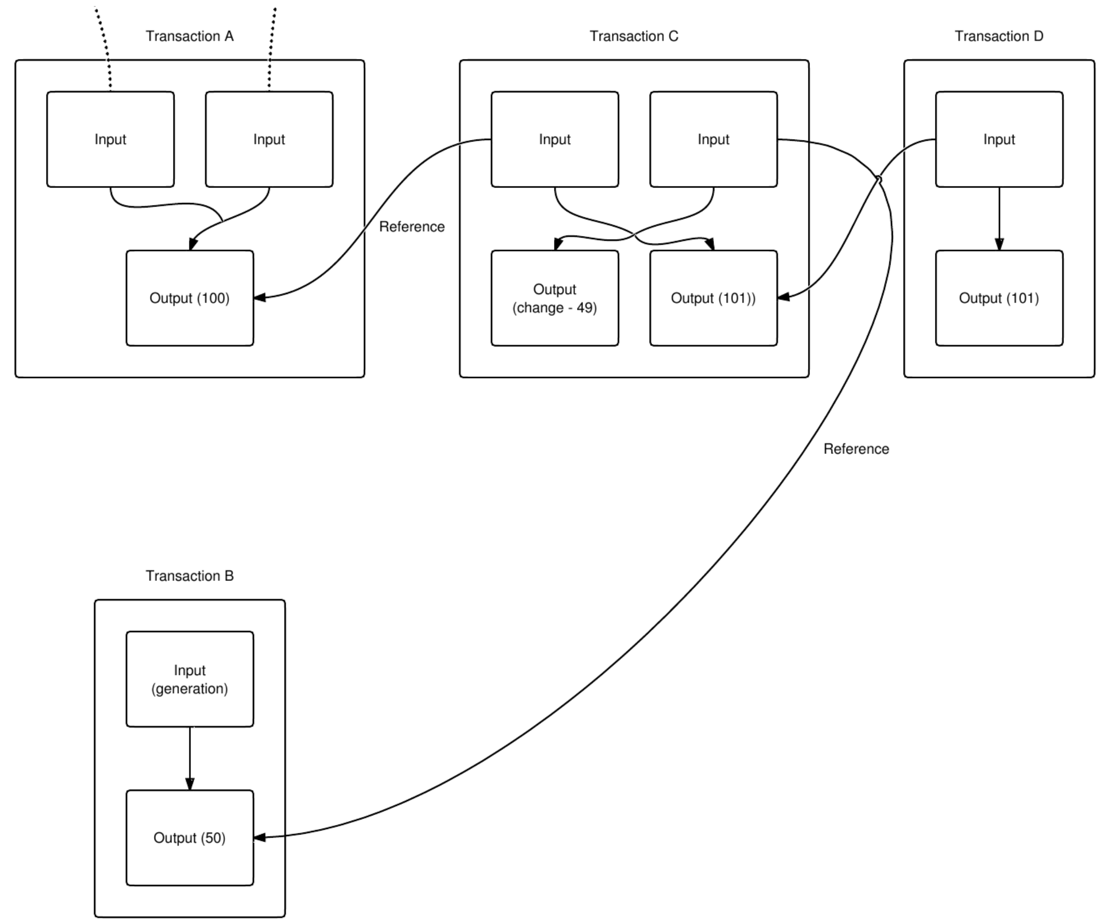


Figure 2.3: 4 example transactions and how inputs are connected to outputs

In simplified terms an output could be seen as the destination of a transaction, in other words it says how much and to whom the transaction is sent to. An input is a reference to a previous output. The inputs take the money from the outputs they reference and that money is used to fun the new outputs.

The inputs and outputs is where Script comes into the picture. Both outputs and inputs contains an incomplete script, together however they complete the script. The script in an output could be seen as a challenge, and the script in the input is the response. When a transaction is tested for validity the input script is appended to the script in the output and is executed. If the script comes out as valid the transaction is also valid. Here is a basic example: Let's say Alice wants to send a transaction to whoever can answer the very complex equation $4 + 3$. Her transaction output would contain the script:

```
4 3 OP_ADD OP_EQUALS
```

If this is executed as is it is invalid. But let's say Bob knows the answer to the equation he can then create a new transaction where the input contains the script:

```
7
```

Just as before this script is not valid by itself. But then the transaction is checked for validity the input will be appended to the start of the output script forming the following:

```
7 4 3 OP_ADD OP_EQUALS
```

Which is a valid script, thus bobs new transaction is also valid and he may spend the money as he see fits.

Obviously most transactions on the Bitcoin blockchain are not this simple. The most common form of transaction contians a script called P2PKH which stands for Pay to public key hash. Before we can go into details on this one however we first need to know about how signatures and sighash work in script and transactions.

2.5.1 Signatures and sighash

Section 2.3 covers public keys and signatures in depth.

Perhaps the most important operation in script is the `OP_CHECKSIG` operation and its cousins. `OP_CHECKSIG` pops two values from the stack, if the script is correctly implemented these two values should be the public key and a signature that was signed with the private key that was used to create the public key.

The question is: what is signed when the signature is created? Broadly speaking it is the hash of the transaction that is trying to spend the output, this is not entirely correct however. Appended to the signature that is a flag called **sighash**. The value of sighash tells the script interpreter what hash was signed during the creation of the signature. There are 4 types of sighash implemented:

SIGHASH_ALL

This can be considered the default sighash, if it is not specified it can safely be assumed that this type was used. This simply means that the entire transaction is signed with all outputs and all other inputs.

SIGHASH_NONE

This one signs the transaction butwithout the outputs, it could be thought of as "I don't care where the money goes"

SIGHASH_SINGLE

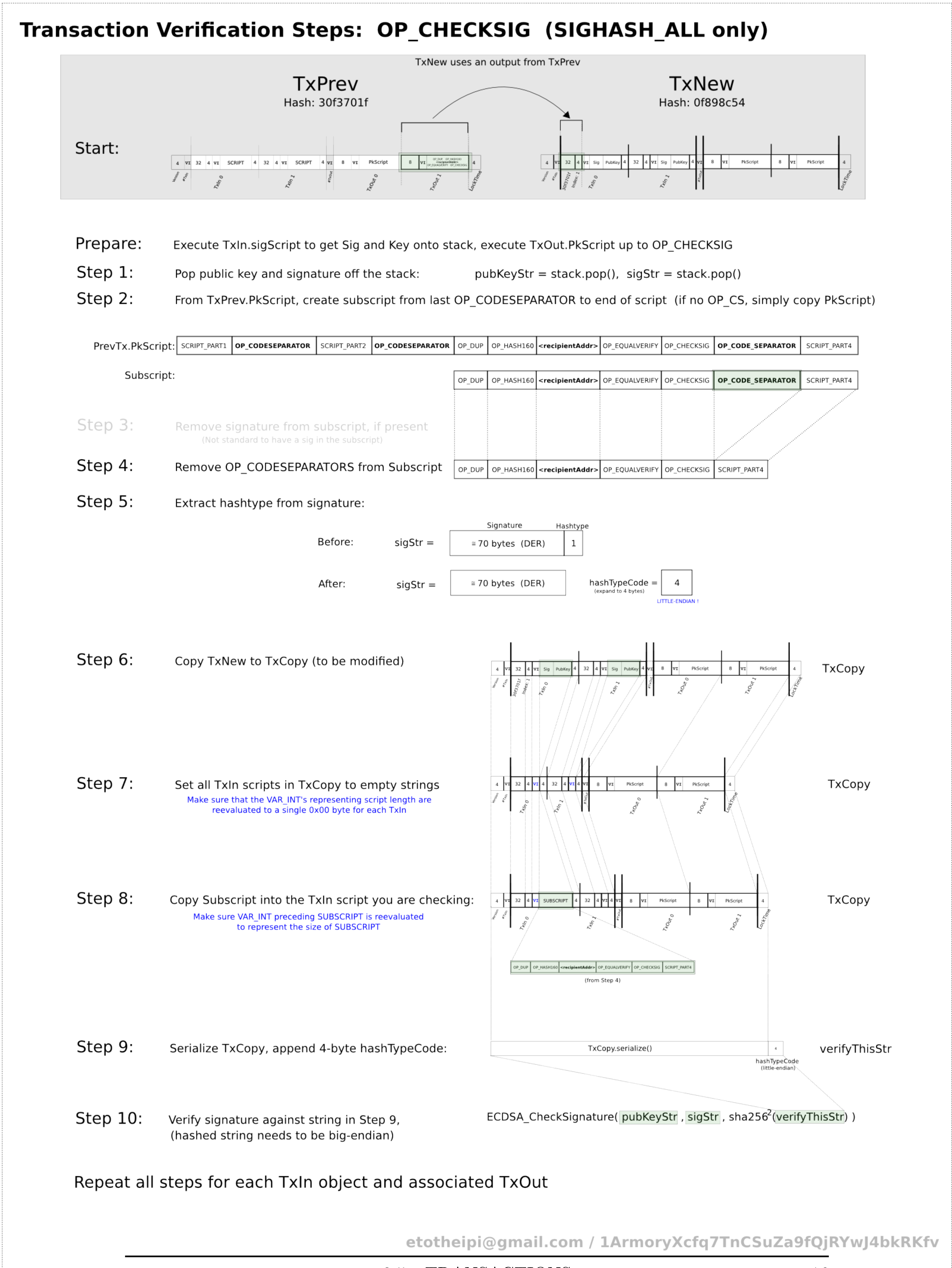
All outputs are removed except the output with the same index as the input that is being signed, then that transaction is signed.

SIGHASH_ANYONECANPAY

Signs the transaction with all the outputs but none of the other inputs. This basically means "The money has to go here, but I don't care if someone else want to fund this transaction also"

More detailed process

On the next page the entire signing process for SIGHASH_ALL is detailed. This is how it is performed in the actual implementation:


Transaction Verification Steps: OP_CHECKSIG (SIGHASH_ALL only)
TxNew uses an output from TxPrev
TxPrev
Hash: 30f3701f
TxNew
Hash: 0f898c54
Start:
Prepare:
Execute TxIn.sigScript to get Sig and Key onto stack, execute TxOut.PkScript up to OP_CHECKSIG
Step 1:
Pop public key and signature off the stack:
pubKeyStr = stack.pop(), sigStr = stack.pop()
Step 2:
From TxPrev.PkScript, create subscript from last OP_CODESEPARATOR to end of script (if no OP_CS, simply copy PkScript)
PrevTx.PkScript:
SCRIPT_PART1
OP_CODESEPARATOR
SCRIPT_PART2
OP_CODESEPARATOR
OP_DUP
OP_HASH160
<recipientAddr>
OP_EQUALVERIFY
OP_CHECKSIG
OP_CODE_SEPARATOR
SCRIPT_PART4
Subscript:
Step 3:
Remove signature from subscript, if present
(Not standard to have a sig in the subscript)
Step 4:
Remove OP_CODESEPARATORS from Subscript
Step 5:
Extract hashtype from signature:
Signature
Hashtype
Before:
sigStr =
≈ 70 bytes (DER)
1
After:
sigStr =
≈ 70 bytes (DER)
hashTypeCode =
4
(expand to 4 bytes)
LITTLE-ENDIAN !
Step 6:
Copy TxNew to TxCopy (to be modified)
TxCopy
Step 7:
Set all TxIn scripts in TxCopy to empty strings
Make sure that the VAR_INT's representing script length are reevaluated to a single 0x00 byte for each TxIn
TxCopy
Step 8:
Copy Subscript into the TxIn script you are checking:
Make sure VAR_INT preceding SUBSCRIPT is reevaluated to represent the size of SUBSCRIPT
SUBSCRIPT
TxCopy
(from Step 4)
Step 9:
Serialize TxCopy, append 4-byte hashTypeCode:
TxCopy.serialize()
hashTypeCode
(little-endian)
verifyThisStr
Step 10:
Verify signature against string in Step 9,
(hashed string needs to be big-endian)
ECDSA_CheckSignature(pubKeyStr , sigStr , sha256²(verifyThisStr))
Repeat all steps for each TxIn object and associated TxOut
etotheipi@gmail.com / 1ArmoryXcfq7TnCSuZa9fQjRYwJ4bkRKfv

2.5.2 Pay to public key hash (P2PKH)

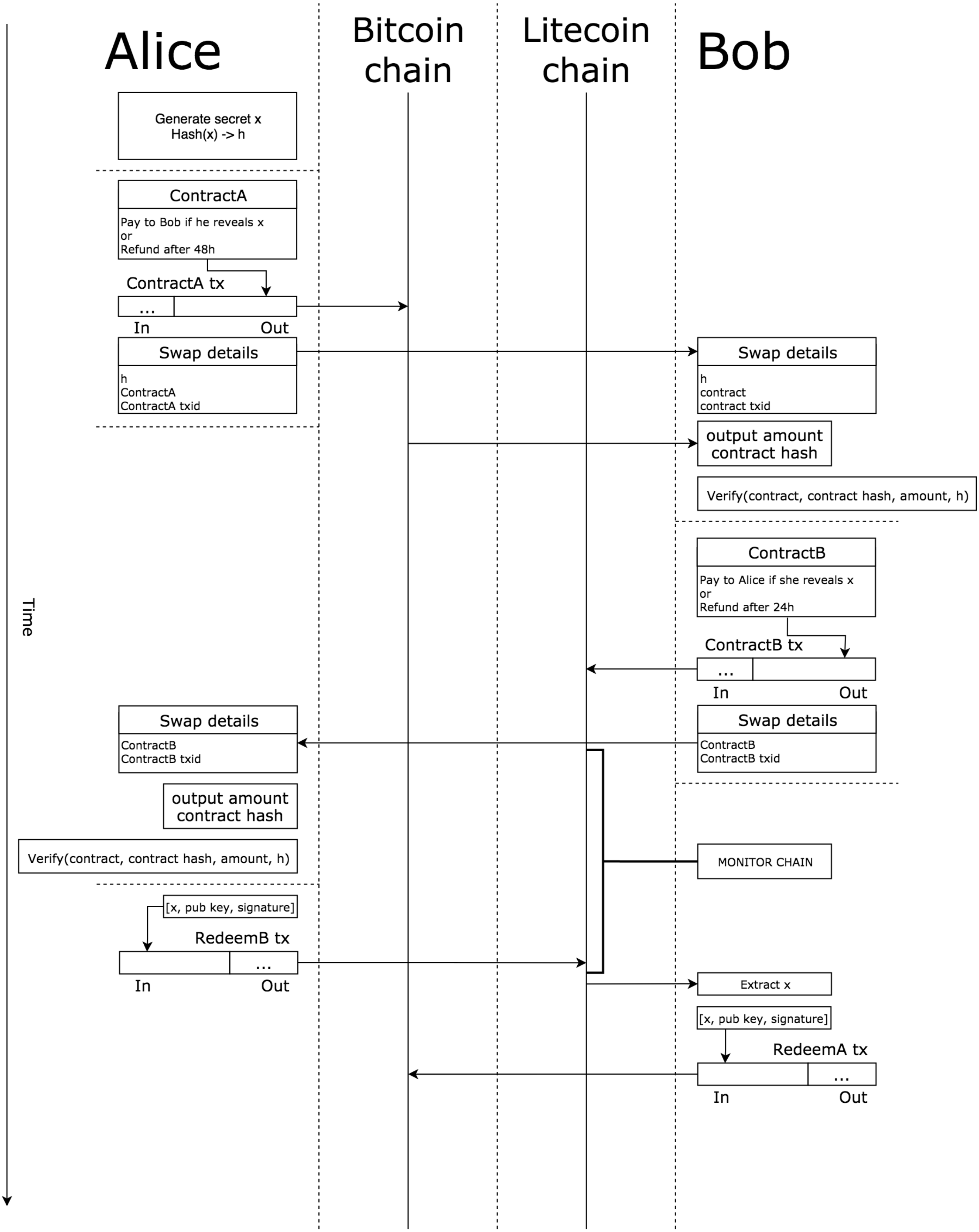
2.5.3 Pay to script hash (P2SH)

2.5.4 Timelock and sequence

2.6 Segregated Witness

2.7 Lightning network

2.8 On-chain Atomic swaps


Alice
Bitcoin chain
Litecoin chain
Bob
Time
Generate secret x
Hash(x) -> h
ContractA
Pay to Bob if he reveals x
or
Refund after 48h
ContractA tx
...
In
Out
Swap details
h
ContractA
ContractA txid
Swap details
h
contract
contract txid
output amount
contract hash
Verify(contract, contract hash, amount, h)
ContractB
Pay to Alice if she reveals x
or
Refund after 24h
ContractB tx
...
In
Out
Swap details
ContractB
ContractB txid
Swap details
ContractB
ContractB txid
output amount
contract hash
Verify(contract, contract hash, amount, h)
MONITOR CHAIN
[x, pub key, signature]
RedeemB tx
...
In
Out
Extract x
[x, pub key, signature]
RedeemA tx
...
In
Out

2.9 Off-chain atomic swaps

Chapter 3

Implementation

Chapter 4

Comparison

Chapter 5

Conclusion & Discussion

Chapter 6

Future research

Bibliography

[1] Genesis block. https://www.blockchain.com/sv/btc/block-height/0.

[2] On scaling decentralized blockchains. https://www.comp.nus.edu.sg/~prateeks/papers/Bitcoin-scaling.pdf.

[3] The original description of atomic swaps on bitcointalk. https://bitcointalk.org/index.php?topic=193281.msg2224949#msg2224949.

[4] A post by satoshi on bitcointalk. https://bitcointalk.org/index.php?topic=234.msg1976#msg1976T.

[5] A survey of bitcoin transaction types.

[6] Text dump of nakamotos last mail. https://pastebin.com/syrmi3ET.

[7] Andrea M. Antonopoulos. *Mastering Bitcoin: programming the open blockchain.* OReilly, 2 edition, 2017.

[8] Zoe Bernard. Everything you need to know about bitcoin, its mysterious origins, and the many alleged identities of its creator. *Business Insider*, Nov 2018.

[9] Adrianne Jeffries. Four years and $100 million later, bitcoin's mysterious creator remains anonymous. *The verge*, May 2013.

[10] Lightningnetwork. lightningnetwork/lnd, Jan 2019.

[11] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2009.

[12] Certicom Research. Sec 1: Elliptic curve cryptography, May 2009.

[13] Certicom Research. Sec 2: Recommended elliptic curve domain parameters, January 2010.