UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN

ORÁCULOS DISTRIBUIDOS EN LA BLOCKCHAIN

TESIS PARA OPTAR AL GRADO DE MAGÍSTER EN CIENCIAS MENCIÓN COMPUTACIÓN

MEMORIA PARA OPTAR AL GRADO DE INGENIERO CIVIL EN COMPUTACIÓN

FRANCISCO JAVIER ANDRÉS MONTOTO MONROY

PROFESOR GUÍA:
ALEJANDRO HEVIA

MIEMBROS DE LA COMISIÓN:
INTEGRANTE 1
INTEGRANTE2
INTEGRANTE3

SANTIAGO DE CHILE
DICIEMBRE 2017

RESUMEN DE LA MEMORIA PARA OPTAR
AL TÍTULO DE
POR: FRANCISCO JAVIER ANDRÉS MONTOTO MONROY
FECHA: DICIEMBRE 2017
PROF. GUÍA: ALEJANDRO HEVIA

ORÁCULOS DISTRIBUIDOS EN LA BLOCKCHAIN

When placing bets we trust the other(s) participant(s) or the institution holding the bet to pay us if we win. In this work we propose a protocol on top of Bitcoin to place bets depending on real world events in a decentralized and trustless fashion. Implementations of decentralized gambling on top of Bitcoin or other cryptocurrencies have been publicized for random events, trying to emulate some casino's game of chance, as the roulette or blackjack. However a distributed solution for non random events is still an open problem.

Our protocol uses of a set "oracles" to bring the outcome of the bet's event into the blockchain. When a certain threshold of the oracles have reported the same outcome for the event we unlock the winner's prize. Protocol was designed to move the money to one of the players in the bet, oracles do not take control of the prize, they only decide between two possible outcomes.

Economic incentives are placed to encourage good behavior of the participants, oracles get paid if and only if they provide the right answer, defined as the answer provided by the majority of them. They are required to place a deposit to participate. This allows the protocol to establish economic penalties if the oracle does not answer or does it wrong.

Once the bet is placed, the prize is locked waiting for the oracles' answer, players lose the control of the money until the oracles decide the winner or a previously defined timeout is exceeded. In which case each player gets half of the prize.

Using Bitcoin scripting language the protocol guarantees fairness and correct execution with a dishonest minority of oracles and a dishonest peer, once the bet is placed. Before the bet is placed a dishonest peer can cause limited economic damage to an honest player. With an attack cost equal to the damage inflicted. Dishonest oracles before the bet is placed can cause small economic damage to both players equally, the cost for this attack is a parameter set by the players.

Una dedicatoria corta. Por ejemplo, A los creadores de U-Campus

Agradecimientos

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1. Gambling

Gambling is the activity of predicting events and placing a wager on the uncertain outcome of those events, with the intent of winning money or valuable goods. A wager can be placed on many different events, in a casino we find randomizing devices as dice, roulette wheels, etc. which are used to get randomized events. In other establishments we can bet on sporting events, such as a horse racing, football games, etc. or the minimum temperature in Santiago during a particular night. Gambling popularity and the large amounts of money at stake inevitably entails a lot of interest in such activities. Most of the time gambling is heavily regulated and taxed, and lotteries are usually owned by the state.

Internet has made it cheaper to open and operate a casino, even without complying with laws of a particular country. This and the massive internet use has moved the gambling industry online[31][17]. The global Internet gambling market was estimated to be worth US$28 billion in 2012 and forecasted to rise to US$49 billion by 2017[16]. However, gambling not only takes place in casinos, lotteries or betting sites, it can also involve two or more individuals with no intermediaries. In Chile, friends usually bet on their favorite football teams.

Of all the different ways for placing a bet, the aforementioned share a common obstacle: participants are required to trust in the other parties to pay if they lose. Even if the bet takes place in a physical casino, where the law can enforce the bet, it is not certain if the casino will be able to pay after the resolution. We might not be aware of the fact, but every time we place a bet we are implicitly trusting in a third party, either the other player or the bet site. For physical casinos this is usually not a problem, as they are regulated by the law, any misconduct can get the casino in legal problems resulting in a revoke revoked. As there is a significant cost involved in starting a physical casino, maintaining a good reputation will attract customers.

Friends usually are trusted people, so trusting them when gambling might not be considered an issue, but the friendship could be put at risk if the bet is not paid. Another option

is to place the bet with the help of a third friend, who helds the money and pass it to the winner when the bet is resolved. Online casinos on the other hand are more problematic as there are many knowns scam schemes, as described by Griffiths [18]. And half of the players at this sites believe the providers are cheating on them [25]. However, some of them are subject to government regulation and many have been in the business for several years. This kind of characteristics could help to indicate thar online site is trustworthy.

But, what if you would like to gamble in a event that no gambling site offers nor any friend is willing to do gamble init? Likely the internet would be the place to look for somebody willing to gamble on this event. Yet, how could you trust the potential person in order to bet with him/her?

1.2. Cryptocurrency

Digital currency refers to any currency stored and transferred electronically. A subset of the digital currencies is called virtual currencies: they are usually defined [3] as a «*unregulated, digital money, which is issued and usually controlled by its developers, and used and accepted among the members of a specific virtual community*».

Based on the interaction of the currency with currencies outside the community there are three types of virtual currencies: The ones with almost no interaction with the outside money, this is usually the case of video games, where its currency is only valuable within the game. A second type is where the currency can be purchased directly using other currency. Here, we observe an unidirectional flow. The third type is when the flow is bi-directional, the users can sell and buy the currency. A cryptocurrency is a bi-directional virtual currency that uses cryptography for security and anti-counterfeiting measures. Virtual currencies are being historically linked to cryptography. The first known investigations [8] to establish a virtual currency were lead by David Chaum, an American cryptographer. However, despite his and others' effort (e-gold[1], Ecash [9], DigiCash, LibertyReserve, among others), virtual currencies never were massively adopted.

By late 2008, a short whitepaper [27] signed using a pseudonym, was released with yet another virtual currency protocol specification. Later in 2009 its implementation was made available as open source code. The main difference with previous implementations was its lack of a central organization, this new coin was completely decentralized. The software started to be run by some early enthusiasts and Bitcoin went from an idea to a usable coin. In the beginning the coins were exchanged for free among the community users. However, at some point the community was big enough and its members started to give value to the coin, then the first exchanges from and to other coins started to take place. Bitcoin transitioned into a bi-directional flow virtual coin.

Then the first online exchanges between bitcoin and other currencies started to appear, and the coin started to gain popularity as people outside the community were able to buy and sell coins. As the money became popular, the idea was taken and a whole generation of

[1] https://www.wired.com/2009/06/e-gold/

cryptocurrencies was born. Today the market capitalization of Bitcoin (this is, the amount of money times its value in USD) is over 25,000,000,000 USD.

1.3. Gambling using Cryptocurrencies

With cryptocurrencies getting more popular, it was only a matter of time until the first sites started to offer some games of chance, acting as online casinos. The only difference of this new sites with a traditional online casino was the currency on which the bet takes place. However, as any other currency online casino, any player who decided to play here is at the mercy of the casino. If the casino does not want or does not have the means to pay, there is nothing the participant can do, and his money is lost. The problems described for online casinos using traditional currencies apply in the same way to the new ones. (More on online casinos at subsection 1.1.)

After some time, people started to see potential of cryptocurrencies at solving some of the trust issues related to gambling. In 2014 Andrychowicz et al. proposed a two-party randomized gambling protocol. Players are not required to trust each other in order to gamble, so even if the loser does not behave correctly, the honest player receives the prize. The protocol is not a representation of a casino game, but effectively allows players to gamble on a random event. Also in 2014, a group of Bitcoin enthusiasts started Orisi[2], a distributed oracles system for cryptocurrency contracts. Orisi allows users to access data from the outside world from the blockchain by using a distributed set of oracles. So instead of trusting in one instance to provide the data, the trust is placed in the majority of several different oracles. More recently, on early 2017, Winsome[3] was released. Advertised as a «*Provably Fair / Trustless Casino*», Winsome is an online casino where wagers are placed in a public smart contract posted in the Ethereum's blockchain. The contract defining the game is enforced by the Ethereum protocol. As May 2017, they do offer two casino games, blackjack and `Rouleth`, which is an online roulette.

Motivated to provide an option to gamble over real world events with untrusted peers. This work proposes a protocol to define the destination of an initial wage between the two players. The decision is made by a set of oracles, which are being paid also within the protocol to behave correctly.

1.4. Objectives

Design and implement a distributed protocol where real world observations can be used as blockchain transaction inputs.

[2]http://orisi.org
[3]https://www.winsome.io

1.4.1. Specific Objectives

1. Provide a protocol to make it possible to gamble with untrusted peers over real world events.
2. Provide the correct economic incentives to the protocol participants to behave correctly, so everyone's incentives are aligned.
3. Implement a proof of concept of the designed protocol.
4. Debate implications and other applications for the designed protocol.

1.5. Methodology

The main phases of this work will be the following:

1. Extensive review of existing proposal and implementations to solve the proposed problem or similar ones. As cryptocurrencies are a recent research field, this review must cover literature as well as community gathering places, such as forums and specialized blogs, magazines, etc..
2. Analysis of current solutions to the problem and similar ones.
3. Design and implementation of a protocol to solve the problem. Implementation is considered very important as the current rate of change of cryptocurrencies is considerably fast, and validating the protocol within a real implementation is critical.
4. Analysis of the economic incentives of the protocol participants, to ensure protocol viability.

1.6. The Protocol

The main idea behind this work is to eliminate most of the single points of trust we can when performing bets. Traditional currencies are produced and controlled by governments, so the first decision was to use a currency without a single controller. We chose Bitcoin mainly for two reasons. It is the first and one of the most stable currencies out there. Changes are made much slower than other currencies, the market backs this claim by making bitcoin the Cryptocurrency with by far the biggest market capitalization. And second, the network supporting bitcoin is much bigger than the ones for other cryptocurrencies. This makes it much harder to attack and take control of the currency.

Our proposed protocol lets users bet against each other over the outcome of future events, without trusting each other nor any single judge. It keeps the money under each player's control and relies on a distributed set of oracles to decide who wins the money.

There are two mains phases in the protocol, the first one is optional and can be replaced at players' will:

1.6.1. Oracles selection

Bitcoin (like most of the cryptocurrencies) includes a scripting language able to control money transferences, well defined and with its execution enforced by the complete bitcoin network. The challenge is to bring data from outside the bitcoin blockchain into it so scripts on it can run on that data. Our protocol relies on several paid "oracles" to bring this data. As the oracles' output will be used to decide who is the bet's winner, it is a crucial step. The chances of a player tampering the list with oracles controlled by him must be minimized. We say this phase is optional as it might be the case both players trust already in a set of oracles.

In this first step the players compile a list of oracles from a distributed and public source, and select randomly from there a subset of oracles to be used in their bet.

1.6.2. Bet resolution

This phase starts after the players agree on the bet and the oracles to be used on it. Both players build and sign a transaction containing all the Bitcoins required for the bet, the bet description and the list of oracles chosen to participate. This transaction is sent to the blockchain to make it publicly visible. So oracles known they are asked to participate. Oracles express their desire to participate by submitting an enrollment transaction into the blockchain. If enough oracles do it, the bet can take place.

Once the required number of oracles are enrolled to participate, the "Bet" transaction is placed in the blockchain, and signed by the oracles participating and both players. All the payments are also set in this transaction: the payments for the winner player; the payments for the oracles that answer properly; and the payments to the players from the oracles that do not behave properly.

Oracles get their payment by answering who the winner is, and as soon as enough[4] oracles vote for one player as the winner, this player can take its winnings.

[4]Threshold defined in the bet parameters, at least $\lfloor \frac{n}{2} \rfloor + 1$ with n the number of oracles.

Chapter 2

Preliminaries

2.1. Hash

Hash is an overload word and it is usually used to define various things:

2.1.1. Hash Function

It is a function able to map data from arbitraty size to data of a fixed size. The range has only elements of a fixed size, so it is bounded by all the elements of that size. If we represent the data in a binary base, the range is bounded by 2^n where n is the size in bits of the output. The domain of the function is unbounded, by the *Pigeonhole Principle:*

$$\exists \mathrm{i}, j \mid f(\mathrm{i}) = f(j), \mathrm{i} \neq j \tag{2.1}$$

When this happen, we call it collision, and for most uses of a Hash function are unwanted.

Hash functions are used for many things: File comparison, instead of comparing files bit to bit, the image of a Hash Function can be compared instead; Hash-Tables, this allows quick lookups for the elements; Finding similar records, by using a Hash Function that produces similar images for similar pre-images, etc..

2.1.2. Image of a Hash Function

If not stated otherwise we will use the word "Hash" to denote the image of some data using a Hash Function.

2.1.3. Cryptographic Hash Function

This refers to a special class of Hash Functions the cryptography has defined to be suitable for its use on cryptographic applications. These functions are designed to be "one way" functions, meaning it is unfeasible[1] to invert if the input to the function is chosen uniformly at random.

In an ideal cryptographic function, the most efficient way to find one of the preimages is by a brute-force search[2]. We call this property *preimage-resistance*. It is also important for this ideal function to be *collision resistat*, this means it is unfeasible to find any two distinct inputs x, x' with the same image, i.e., such that $h(x) = h(x')$.

When using this ideal function, on average, producing a (second) preimage requires 2^n operations, and producing a collision requires at least $2^{n/2}$ operations [30].

2.2. Digital Signatures

The idea of "Digital Signature" was introduced in 1976 by Diffie and Hellman in "New Directions in Cryptography"[12]. This work introduced what they called "Public Key Cryptosystem", where enciphering and deciphering operations use different keys, E and D, such that computing D from E is computationally infeasible. Today this pair is widely used and known as Public Key (PK) and Secret Key (SK).

The public key cryptosystem, or asymmetrical cryptography was created to solve one important problem of symmetrical systems[3]: It is imposible to start a secured communication in an insecure channel without previously exhange of a key using a secure channel. To stablish secure communication within an insecure channel participants makes its PK publicly available to the others. Anyone willing to talk to another participant must cipher its message using the public key of the receiver, this way the only one able to decipher the message is the intended receiver.

A digital signature, as its name indicates, is a mechanism to provide protection against third party forgeries. It must be easy for anyone to recognize as authentic, but impossible for anyone but the signer to produce it. This is especially challenging since any digital signal can be easily copied.

It works within the public key cryptosystem. The signer uses its SK to produce a signature over the message to sign, and anyone with the signer PK and the message can determine the validity of the signature.

The most important property of a digital signature is that it does not matter how many

[1] We say something is computationally unfeasible when even it is computable, it will require far too many resources to do it.

[2] Also known as exhaustive search, consisting of enumerating all the potential solutions and checking which of them satisfies the predicate

[3] As opposed to the asymmetrical one, this system uses the same key to cipher and decipher the messages.

pairs <message, signature> a third party has seen, but it does not make it easier to generate a signature for a new message.

2.3. Ecash

Digital currencies have been a research topic since at least 1983 when David Chaum [8] introduced Blind Signatures. A form of digital signature where the content of the signed message is blinded, so the entity signing the message do not get to see it. This technique was used to provide untraceable payments in a cash system where however anybody can check the signature is valid.

The field has been an active topic in the academy and as business intent. Much research has been published proposing new schemes and cryptographic primitives [29][7] [6][1][24].

In 1990 David Chaum founded DigiCash, which developed an early electronic payment based on blind signatures. Payments using the software were untraceable by the issuing bank or any third party, including the government. However, the company was not able to beat credit cards in the electronic commerce and files its bankruptcy in 1998.

In 1996 e-gold allowed its user to buy electronic money ("grams of gold"[4] that were backed by precious metals held by the company [20]. The users can buy, sell and transfer the ownership of the metals over the Internet. In 1999 the *Financial Times* described e-gold as the only electronic currency that has achieved critical mass on the web. However, its success contributed to its demise. It was used for fraud, phishing, cyber crime gangs, etc.. Law enforcement agencies began to characterize e-gold as the favorite payment system for criminals and terrorists[5]. By 2007 the justice started to seize e-gold balances that ended up with the suspension of the service.

Other initiatives suffered the same fate. Closed by the governments or for lack of interest are the reason for the failure. New cryptocurrencies are virtually impossible to close by a government because its distributed design, and so far people have been very enthusiastic about them. This might signify they will prevail much longer than previous solutions.

2.4. Bitcoin

Bitcoin is the first fully distributed cryptocurrency made publicly available, proposed in 2008 by Satoshi Nakamoto (a pseudonym) [27]. The same author shared as open source code a implementation of the protocol in January 2009. And the protocol has been running ever since.

Nevertheless, Bitcoin is not the first idea of electronic cash. The idea of electronic cash

[4] Also of platinum, silver, etc..

[5] "Feds out to bust up 24-karat Web worry". NY Daily News. 2007-06-03. Retrieved 2017-07-27

has been present within the cryptographic community since at least 1983, when Chaum [8] proposed a system for anonymous payments. And the attempts kept going for another three decades, amd hundreds of papers have been published with improvements of e-cash schemes [4]. Why is Bitcoin so popular and why has it achieved the notority that three decades of academic research on the field could not achieve?

Barber et al.[4] suggest a few key points to explain why Bitcoin was the first electronic currency to take off.

1. No central point of trust. Bitcoin is a fully distributed system, there are no trusted entities in the system. The only asumption is that the majority of the network participants are honest. Every previous proposal had a central trusted entity for critical tasks, as preventing double spending and coin issuance.
2. Predictable money supply. The money supply is minted at a defined and transparent rate, defined from the begining of the protocol.
3. Transaction irreversibility. Bitcoin transactions quickly become irreversible. This is a big difference with credit cards, where chargebacks have been used largely to commit fraud.

The main technical advance in Bitcoin is its database, the `blockchain` [14][28]. The blockchain is a distributed database formed by an always growing list of blocks, where each block contains the data to be stored, a timestamp and a link to a previous block. Its fully distributed nature allows bitcoin to lack a central authority.

2.4.1. Transaction

Bitcoin works with accounts where coins are stored, these accounts are identified with an address[6], for this reason sometimes both words are used interchangeably. The address is not private, and people share theirs when willing to receive money.

The address represents a hash of a public key. Therefore, the owner of the account is however control the private key of the address. Bitcoin uses Elliptic Curve Digital Signature Algorithm (ECDSA)[7] to ensure the owner of an address is the only one able to spend its content. Getting a new address is free, it only requires one to generate a random byte string and get a ECDSA private key from it. ECDSA allows the public key derivation from the private key. By hashing the public key the address is then obtained. Many libraries and most of the bitcoin wallets implement the algorithms to generate new addresses.

A transaction is the only way to move bitcoins from one account to other one, it is basically two lists: a list of accounts where the money is pulled from, called inputs; and a list of accounts where the the bitcoins are going to, a simplified view in the figure 2.1.

[6]In the wire, an address is a 25-byte value, for human consumption we usually see it in its encoded representation of base 58, resulting in a string of 25-33 characters.

[7]ECDSA is a digital signature algorithm using Elliptic Curve Cryptography. It is an asymmetric scheme, with a private and public key. Bitcoin transactions are secured with a private key signature and validated using the public one.

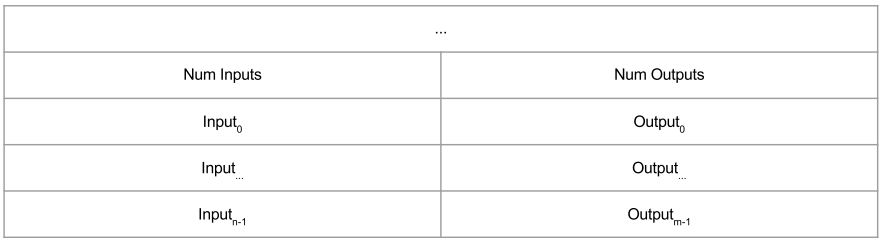

Figure 2.1: Simplified Transaction

There is only one exception to this rule; the miner that builds each block is allowed to create a transaction without intputs and send money to his account. This especial transaction called generation transaction and its amount is defined in the protocol. This is the only way bitcoins are generated.

A transaction input points to a previous unspent output and proves it has the right to spend that output. An output contains[8] the address of the account it is transferring the money to, so any input signed using the private key of that address has the right to spend the output. This implies that the money from an account must be spent in the same amount the money was received. There is no way to spend just a fraction of the money received in a previous output. If an output of $\$10 BTC$ is received, when trying to spend it, the same amount must be spent. If willing to spend just a portion, a second output is created and sent to the same account.

2.4.2. Blockchain

It works as the bitcoin's ledger keeping record of all transactions and coin generation that had ever taken place in the protocol. It is completely distributed, public, and anybody can participate and get a copy of it. This makes it simple to prevent double spending while being sure the received coins are valid, as anybody can examine where each coin came from.

As any other distributed system, the blockchain must resolve the consesus problem [15]. Get all the participants to agree on the data. This is a fundamental problem to any distributed system. In the the blockchain anybody with an internet connection can be part of the protocol, so solving this problem is quite challenging. Some authors argue that the blockchain is the first practical solution to the Byzantine Consensus problem [26] [32].

Proof of work is the algorithm used by the bitcoin blockchain to seek consensus. Each entity trying to add data to the database must prove it has done some required work. This algotithm was designed originally to fight email spam by requiring the sender of an email to prove that a small job was done in order to send the email [13]. It works by using a hard to calculate, but easy to check function. This way the receiver or the mail server can easily check if the sender did the required work. However this work was much harder. The difficulty of particular work is defined by the amount of computational power required to get it done.

[8]This is a simplification, not every output works this way. Details are in subsection 2.4.3.

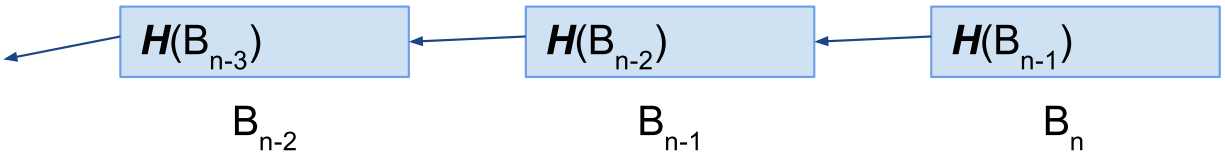

Figure 2.2: Blocks linked to each other in the blockchain.

The atomical piece in the blockchain is the block. Each valid block carries transactions of the protocol and a proof of work. Therefore every entity trying to get a valid block into the database needs to collect transactions and solve the puzzle to get a valid proof of work for its block. This process is called mining, therefore the entities trying to get a valid block are called miners. A block is linked to the previous one, as show in the figure 2.2.

The mining process is like playing the lottery: tickets are distributed among every miner until someone gets the winning one. The number of tickets each miner gets is proportional to the work he/she is doing, so a miner doing more work will have a bigger chance at winning the lottery and mine a block. However, any miner can win.

Of course, in the real implementation there are no tickets issued to the miners. Proof of work consists in building a block with a hash under a threshold value, so the miners should reorder and change the block until the hash fulfills the requirement. There is no known algorithm to do this in a better way than brute force, so the only method to get a hash that mets the criteria is to try with different block configurations. In the block there are also some bytes of nonce, a timestamp and transactions to be changed to get different hashes.

Once block is produced, all the other miners need to delete the transactions added by the block from the one they are building and update the link to the new last block. And they start to mine a new block. By design a block must be produced every 10 minutes, so the work required to mine a block is adjusted periodically to meet this goal.

The structure of a Bitcoin block is shown in figure 2.3, the fields with the gray background represents the block header, the data hashed to get the block's hash. The transactions are indirectly hashed in the Merkle Root[9].

As expected in a protocol with many participants, there are times where more than one block is generated with the same parent (figure 2.4). This is called "fork".

In order to achieve consensus, the protocol determines that the chain with more work[10] on it is the active chain. So when a fork happens there are two active chains, while having a non unique active chain miners will try to mine in any of the candidates with the same work. A block mined on one of the branches will decide which is the active one because it adds more work to the chain. However the situation that originated the fork can repeat itself and prevent having one consensus branch, which is very unlikely [11] to happen during a long

[9] A `Merkle Tree` is a tree in which each non leaf node is labeled with the hash of its children's labels. In the block each transaction is mapped into a tree leaf. So the root of this tree hashes all the transactions.

[10] The amount of work in a chain is the sum of the difficulty of every block on it.

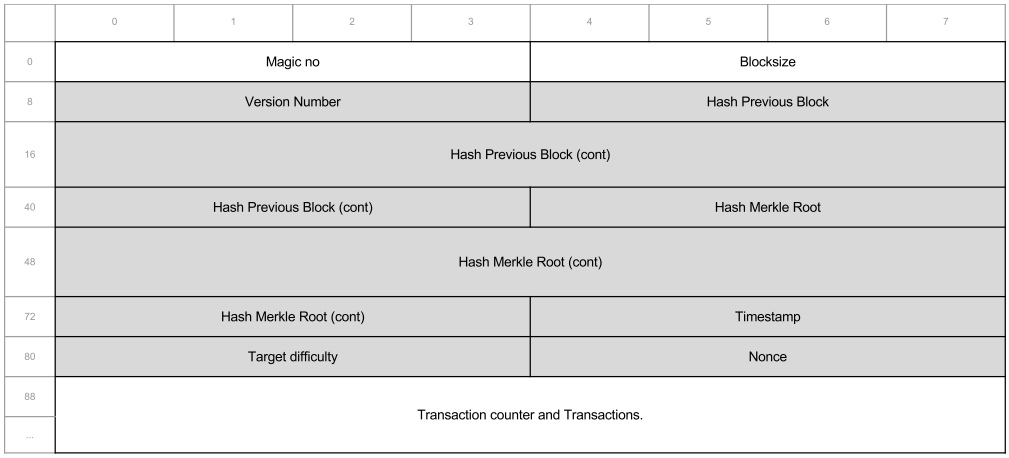

Figure 2.3: Block Structure

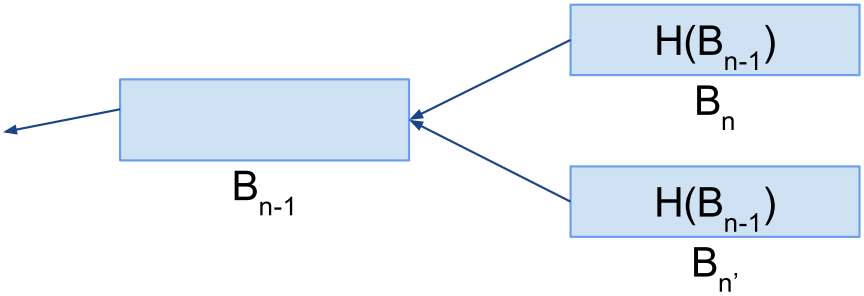

Figure 2.4: A fork in the blockchain.

time.

The chain structure gives a chronological order to the transactions in the protocol, so it makes easy to check if a transaction is valid. Any participant willing to probe the validity of a given transaction needs to evaluate the script (see section 2.4.3), and check the block where the output being spent is stored up to the current block and see if the money was already spent in a different transaction.

2.4.3. Script

When sending money, there is a little more than we saw at section 2.4.1. In an input (figure 2.5) there is more than a signature, and at each output (figure 2.6) also more than an address.

An output does not send money to a given address, but defines how the money can be spent. Currently there are two formats in use. The most used is called "Pay To Public Key Hash" (P2PKH)[11]. The other is called "Pay To Script Hash" (P2SH).

[11]The key hash is the address of an account

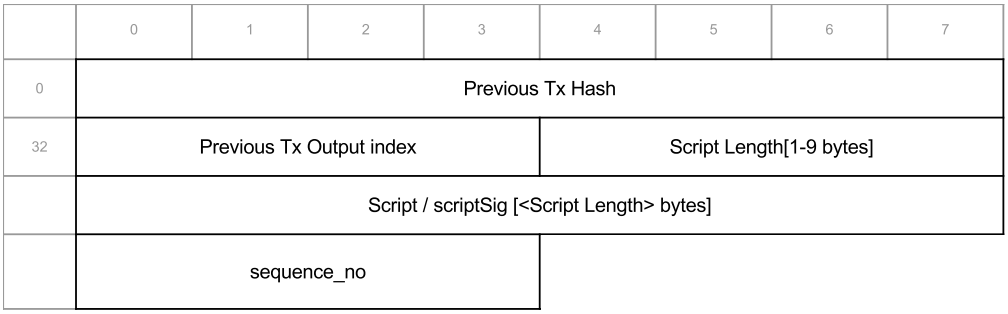

Figure 2.5: Wire format of an Input.

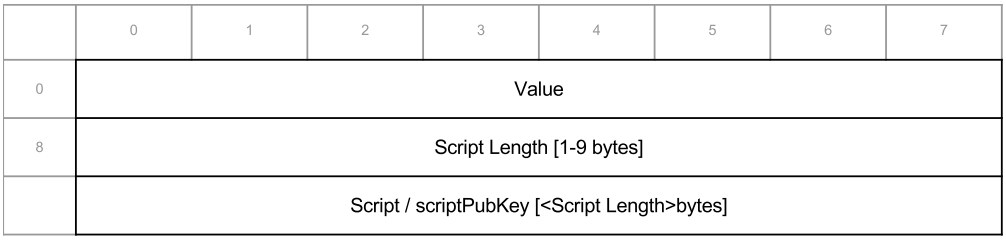

Figure 2.6: Wire format of an Output.

As figure 2.6 shows, the output has a script on its wire representation. This script is written in a small stack based language. It is read from left to right and it is purposefully not Turing-complete. The script is evaluated using the scriptSig as input. If the transaction willing to spend this output provides a valid[12] scriptSig, the output is available to be spend. This is how a P2PKH script looks like:

```
OP_DUP OP_HASH160 <pubKeyHash> OP_EQUALVERIFY OP_CHECKSIG
```

It receives two values as input: <pubKeyHash> and <sig>. And the execution is explained in table 2.1.

The scriptPubKey in a pay to script hash transaction is even simpler:

```
OP_HASH160 <scriptHash> OP_EQUAL
```

This script is pretty simple, it takes the first element of the stack, calculates its hash and compares it with '<scriptHash>'. The first element in the stack is however a complete script, and after checking it hashes to the expected scriptHash. It will be evaluated with its required input. This implies the scriptSig now holds the script and its signature:

```
<sig> >script>
```

A execution of a sample P2SH is shown in table 2.2.

A P2SH transaction allows different conditions to redeem its outputs. A complete list of the operations supported by the Bitcoin scripting language can be found at the Bitcoin wiki: `https://en.bitcoin.it/wiki/Script`.

[12] A script is considered valid if after its execution the value in the top of the stack is True.

Stack	Script
Step 1: *Constants from scriptSig are copied to the stack.*	
<pubKey> <sig>	OP_DUP OP_HASH160 <pubKeyHash> OP_EQUALVERIFY OP_CHECKSIG
Step 2: *OP_DUP copies the top element from the stack.*	
<pubKey> <pubKey> <sig>	OP_HASH160 <pubKeyHash> OP_EQUALVERIFY OP_CHECKSIG
Step 3: *The hash of the top element is calculated.*	
H(<pubKey>) <pubKey> <sig>	<pubKeyHash> OP_EQUALVERIFY OP_CHECKSIG
Step 4: *The destination address is moved to the stack.*	
<pubKeyHash> H(<pubKey>) <pubKey> <sig>	OP_EQUALVERIFY OP_CHECKSIG
Step 5: *The destination address is compared with the Hash of the Public Key (PK) provided by the sig Script. This checks that the provided Public Key is the one from the intended receiver.*	
<pubKey> <sig>	OP_CHECKSIG
Step 6: *Using the already verified PK, the script checks that the transaction was signed using the corresponding Private Key. This step secures the transaction from tampering an proves it was sent by the private Key controller.*	
True	

Table 2.1: Script evaluation to check a P2PKH transaction.

Stack	Script
Step 1: *The serialized script and its input are copied to the stack.*	
{<pubkey> OP_CHECKSIG} <sig>	OP_HASH160 <scriptHash> OP_EQUAL
Step 2: *The serialized script hash is calculated.*	
H({<pubkey> OP_CHECKSIG}) <sig>	<pubkeyHash> OP_EQUAL
Step 3: *The expected hash is pushed to the stack.*	
<pubKeyHash> H({<pubkey> OP_CHECKSIG}) <sig>	OP_EQUAL
Step 4: *Hashes are compared, and if they match, the script is deserialized and evaluated.*	
<sig>	<pubkey> OP_CHECKSIG
Step 5: *The public key is pushed to the stack.*	
<pubKey> <sig>	OP_CHECKSIG
Step 6: *The provided signature is validated using the public key.*	
True	

Table 2.2: Script evaluation of a P2SH transaction.

2.5. Previous Work

There are several attempts to provide information to the blockchain from the outside. By the way they gather the data we divide them in "Distributed Oracles", where the data is gathered by a group of third party participants. Or as "Data Feeds", where the data is provided by a centralized party, using some techniques to authenticate the data.

We also add "Trustless Distributed Casino" as they provide a solution to a similar problem. The perform trustless bet, but not on real world events, it is limited to bets on random events.

2.5.1. Distributed oracles

Orisi

Orisi [23] is a distributed system for bitcoin smart contracts that relies on multiple oracles to bring information from outside of the blockchain. It allows its users to transfer money from one address to another when a condition is met.

In the first step both players need to chose 7 oracles, them will decide the bet winner. They are chosen from "The Oracle List", a curated list with oracles. A multisignature address

is then generated to store the money while the bet takes place. A multisignature address is defined by `m` addresses and a required number `n` (`n` < `m`) of them to sign. A valid signature for a multisignature address is generated by using at least `n` out of the `m` addresses defining it.

The multisignature address generated will store the money until the oracles decide where the transaction goes. To avoid the oracles sending the money to themselves the multisignature transaction include the address of the receiver, so we want a $1 + (n \text{ of } m)$, where the extra signature is from the receiver. As this kind of transaction is not considered standard[13], Orisi uses a bigest multisignature address, where instead of using n out of m oracles, it adds more receiver keys. Requiring $m + 1$ signatures of $2m$ - $n + 1$. With this configuration the oracles are not able to move the money by themselves, and at least one signature from the receiver is required.

2.5.2. Trustless distributed casino

Winsome.io

In may 2016 Rouleth [21] was launched as a distributed application on the ethereum network. Offering its players a "provably-fair", real money roulette. "BlockJack" was launched in early 2017 using the Ethereum network. It is the first playable blackjack game on the Ethereum mainnet.

Winsome.io is the instance where these games are enclosed and offers unique advantages over traditional casinos (physical and virtual), like no need to trust on it, and complete control over the funds the entire time while playing. It does work in a distributed fashion using smart contracts, publicly availables for everyone's scrutiny.

Winsome.io provides its users trustless gambling over random events. By using the ethereum network as backend. It has been quite successful and one of the most popular descentralized applications on the Ethereum Network.

2.5.3. Secured data feeds

Oraclize

Oraclize [22] provides an interface for using data fetched from a web site in the ethereum blockchain, working with arbitrary URLs or queries in certain web services, as "Wolfram Alpha"[14]. It provides an Authenticity Proof of the data gathered, so the user can check that

[13]Non standard is recognized as a valid transaction by everyone, however by the time this article was written only about the 5% of the mining power will mine it. Including this transaction in the blockchain will take on average more time than a standard one.

[14]Wolfram Alpha is a knowledge engine able to answer queries rather than provide links to data sources, as a search engine does.

the data provided by the interface was generated by the source and had not been tampered.

Town Crier

Town Crier [34] is an authenticated data feed system for the ethereum blockchain. As oraclize it works as a bridge between web feeds, and the blockchain. It uses an Intel technology called "Software Guard Extensions" [10], than provides some execution guarantees of the software executed by hardware protected areas. This protects the execution of the data feed even with the the host OS, BIOS or any other software in the machine compromised.

Chapter 3

The Protocol

3.1. Overview

We split the protocol to make clear which part of it is optional and can be skipped if players can agree in a set of oracles. In this subsection we present an overview before the full explanation of the protocol.

3.1.1. First part: Oracle selection

1. The first step is to compile a list of available oracles, we use the blockchain as decentralized database. Everyone willing to be an oracle can send a transaction to register into the blockchain. Players can compile the list from all registered oracles or might apply some filter; eg time of registration or if the oracle already participate in a previous bet.
2. Players negotiate the bet parameters required for this step, as the number of oracles to use.
3. In order to decide which oracles to use, the oracles need to pick a subset of the list compiled, they do this by running a distributed coin tossing protocol. With this, they can be sure the list generated is a random subset of the full list. If the list is big enough, the chance of one user controlling the oracles gets smaller. As it would be too expensive to control almost all the oracles in the list.

3.1.2. Second part: The Bet

1. Players negotiate bet parameters as the fees to the oracles, the bet resolution timeout, the amount of money to gamble, the winner on each outcome of the event, the oracles penalties on misbehavior, etc..
2. Players build, sign and send a transaction to the blockchain with the bet description, including the IDs of the oracles they want to decide the winner. We call this transaction "Bet promise", as the players commit to the bet by placing it. The wage is also on it.

The other purpose of this transaction is to invite the oracles to participate in the bet, we make its ID public so they can identify itself and enroll to participate as oracles.

3. The oracles will see the transaction asking them to participate in the bet, they will evaluate it and, if they are interested. They will reply with a transaction containing a reference to the "Bet promise" transacation and a deposit as commitment that they will participate in the process.
4. When the players see the answer from the expected number of oracles, they will send the "Bet" transaction with funds of the bet and the oracles' reward. If not enough oracles reply to the call, a second invitation can be send to a different set of oracles to fill the available spots.
5. As soon as the bet event takes place, oracles are able to collect its payment from the Bet transaction. This payment gets available by making public, -voting- by the winner. After the threshold number of oracles collect its payment, the winner player is able to collect its prize, its private key and the oracle votes are required to get it.
6. After a second timeout, players can take the deposit from the oracles that did not participate in the bet resolution.

3.2. Oracles

The first issue to solve when making decisions over events in the world is to define who track and define the outcome of said event. In our day to day we get information about events from a variety of sources. The television, an internet portal, our eyes among many others. Any protocol willing to make decisions over events needs a source for those events.

In order to keep the protocol decentralized we define its data source as a set of entities, called oracles. The decision is made by the oracles voting on the outcome of the event. In this scheme the decision does not rely on a centralized entity, but in a group of them.

Oracles are rewarded when provide the correct answer to resolve the bet. We define the correct answer as the one gave by at least m of the n oracles where $\lfloor \frac{n}{2} \rfloor < m$. When providing the incorrect anwswer oracles do not get paid, and when giving both answer they get penalized because its misbehavior. This gives strong economic incentives to the oracles to answer as they expect the other ones are going to answer. A discussion on how this incentives influence oracles behavior is available in subsection 4.1.2.

3.3. Players

Players are the ones wagering the money, within this work we define its number to 2, the idea behind this work can be extended to more than two players, however the code in the transactions gets more complicated, so transactions increase on size, this makes the protocol more expensive.

Players participate in the protocol with the expectation to predict the outcome of the event and win some money, they might not trust each other but they collaborate to build the transaction of this protocol, expecting to win. The protocol lock the players' money before the event they are wagering on takes and let the oracles decide where the money goes, this prevent the losing player from not paying the bet. As players are requesting the oracles' services they pay for it and the cost of transactions required in the protocol.

3.4. The Protocol

3.4.1. Notation

Before explaining the protocol we introduce the following notation, based in the work of Andrychowicz et al. [2], to describe the transactions.

Table 3.1 represents a transaction that spends two outputs from two different previous transaction and creates three outputs:

T_x (**in:** T_{y1},T_{y2})	
Inputs:	
$T_{y1}[\alpha]$	σ_1
$T_{y2}[\beta]$	σ_2
Outputs:	
(ν_1)	γ_1
(ν_2)	γ_2
(ν_3)	γ_3

Table 3.1: Transaction notation example.

- T_x: Transaction's name/id.
- T_{y1},T_{y2}: Transactions being spent.
- α, β: Indexes of the output being spent, this is the position the outputs appear in its transaction, starting from 0.
- σ_1, σ_2: Input script, values required to satisfy the output script being spent.
- ν_1, ν_2, ν_3: Value of the output.
- $\gamma_1, \gamma_2, \gamma_3$: Outputs' script, need to be satisfied in order to spend its corresponding output.

For the scripts we use the logical conjunction ($\wedge$) when both propositions must be satisfied. Logical disjunction ($\vee$) when we need to satisfy only one of the propositions. Signatures always sign its own transaction, when a signature from the participant A is required, we say: $\mathcal{S}_A$. Sometime a literal is required, usually a preimage of a known hash ($H(l)$), we use just l. Some paths of the transaction are unreachable before a defined time, we use τ_k to disable

the path this expressions is in before the time k.

For instance, the script at 3.1 requires to be evaluated after the time k and a signature of U. Or l and a signature of U.

$$\mathcal{S}_U \wedge (l \vee \tau_k) \tag{3.1}$$

3.4.2. First part: Oracle Selection

The oracles are a key piece in the protocol, as they get to decide who gets the prize money. The first part of the protocol defines a way to select them in a trustless way. The idea is quite simple, players selects from a list of oracles a subset to participate on its bet. With a big enough list, selecting randomly from it reduces the chances from any of the participants to influence on the selection.

Oracle list

In order to get a trustworthy list, we define a few key properties: It must be a decentralized list; anybody willing to be an oracle can enroll itself; and must be visible for both of the players.

As we saw in sub section 2.4.2, we already have a public distributed database to store information. We use the blockchain to keep the list of oracles, this provides tampering protection, a public database and a distributed source for the list. In order to let anybody enroll to be an oracle, the enrollment is a simple transaction generated by the oracle and sent to the blockchain. We defined our own string as protocol indentification, but different protocols can use different strings to define other lists.

Oracle registration

$T_{Registration}$ (**in:** $T_{oraclepreviousstx}$)	
Inputs:	
$T_{Oraclepreviousstx}[\alpha]$	σ
Outputs:	
(ν)	γ
(0)	OP_RETURN {ENROLLMENT DATA}

Table 3.2: Oracle Registration.

In this registration transaction the oracle takes money from a controlled unspent output and returns all the money where it wants in the first output. The registration happens with the second output. OP_RETURNS makes this output invalid, so nobody can spend it. Which don't matter that much, as the value is zero. But the operation allows us to insert

arbitrary data, and it's here where we enter a defined string saying the oracle's address and its will to participate as oracle.

There is no required deposit for registration, however the transaction fee must be paid when sending the transaction. Some may argue than a higher price to register an oracle will decrease the chances of an individual controlling the majority of the list. If that is the case, increasing the cost by adding a required a unspendable output does not require any change in the transaction 3.2, as the unspendable output already exist. Adding a deposit spendable by an address will require a new output, but the idea remain the same.

When this transaction is submitted to the blockchain, players can look into this transaction and recognize it as an oracle registration.

Compiling oracle list

There are a few parameters players must agree in order to select oracles from the blockchain list. First they decide the period of time[1] they will consider oracles from. Some participants might want to avoid recently registered oracles, as they might have an higher chance to be controlled by the other player. Others might argue too old oracles are likely to be inactive, in order to avoid oracles registered long time ago.

Second, they decide the list to get the oracles from, and if they want to filter out oracles, for example they might decide to exclude oracles that paid less than b bitcoins on fees at registration time. Finally they decide the number of oracles to use, also they can decide to select a few more than the required oracles, anticipating one or more of the selected oracles will not reply to the invitation.

Once they decide the filter and which blocks to use for retrieve the oracles, both players can compile the same list of availables oracles. Hashing the list and compare the hashes helps the users to be sure they had compiled the same list. This list is the source for selecting the oracles to use, players just need to decide which of them to use.

Oracle selection

If this list is big and there is a cost to enroll on it, a random selection from it decreases the chance for any player to get a possible controlled oracle into the final list. And decreases even more to control the majority of them, required to decide the bet.

As trust in the other player is not required for this protocol, both players need to have the certainty the election from the list is random. In order to achieve this property we use a protocol originally proposed to flip coins over the phone [5]. Today this algorithm is mostly known as "Coin Tossing", and lives in a subfield of cryptography called Multi Party Computation. Multi party computation, or secure multi party computation aims to provide

[1] Measured as a range of blocks in the blockchain.

protocols for computing public functions and gets its results while participants keeps their input private.

The idea of the Coin Tossing we use is to get a random bit, as neither of the players trust the other to select the bit randomly, both players select a bit and the they XOR it with the other one. This way, does not matter how the other bit was chosen, the result is random. There is one important restriction when using this protocol, the bit must be chosen before knowing the value of the other, otherwise if one bit is known the second one can be selected in order to get the desired outcome. If we were physically together we would write down the bit in a paper, wait for the other player to write his and then reveal both bits and perform the XOR. However we would like to run this algorithm through the phone or in this case the computer. The idea is the same, but instead of writting down into a paper, players "commit" to the value they just chose randomly by sending to the other player a "commitment". This commitment binds the player to the value calculated, without revealing it. Once both players receive the other's commitment, they send the bit they chose. They check the received value against the previously received commitment, and if they match, the protocol outputs a random bit. Otherwise, a player tried to cheat and the protocol aborts as there is no way to calculate a random bit.

If we have a list both players agree with, and we can also produce random bits, selecting a number of oracles from the list is a trivial exercise. After this step, players had decide the oracles to use. We represent the total number of oracles participating in the protocol by a n, and the required number to decide the bet as m, where $\frac{n}{2} < m$.

3.4.3. Second part: The bet

Bet Promise

Once players decided the oracles to use, whether using the *Oracle Selection* part or by any other mean. They need to agree in the terms of the bet, the event and who is the winner on each outcome, the time available for the oracles to answer, money required by each player, fees of the oracles, deposit required to the oracles, etc.. Once all the bet parameters are set, they are serialized and its hash is calculated, players puts all the money required to run the protocol, including fees and the prize into a transaction. The selected oracles, bet hash and a method[2] to get the transaction full description are appended to this transaction in plain text and the transaction is sent to the blockchain. We call this transaction "*Bet Promise*", as it is a commitment from both players to the bet:

[2]For instance an URL to a website with the bet description. Oracles are responsible to check the description fetched with the hash provided in the transaction.

$T_{BetPromise}$ (**in:** $T_{Aprevioustx}$,$T_{Bprevioustx}$)	
Inputs:	
$T_{Aprevioustx}[\alpha_A]$	σ_A
$T_{Bprevioustx}[\alpha_B]$	σ_B
Outputs:	
(0)	OP_RETURN {Bet Channel, Oracle List, Bet hash}
$(\mathcal{P} - \mathcal{F}_{BetPromise}/2)$	$(\mathcal{S}_A \wedge \mathcal{S}_B)$ $\vee$ $(\tau_{Bet} \wedge \mathcal{S}_A)$
$(\mathcal{P} - \mathcal{F}_{BetPromise}/2)$	$(\mathcal{S}_A \wedge \mathcal{S}_B)$ $\vee$ $(\tau_{Bet} \wedge \mathcal{S}_B)$
(c/n)	$(\mathcal{S}_A \wedge \mathcal{S}_B)$
$(\dots)$	$\dots$ ‹n - 1› $\dots$

Table 3.3: Bet Promise

The inputs for this transaction are not relevant to the protocol, they spend money from both player to pay for this bet. There can be more than one input by player, and change outputs if required. As it's unlikely the players have an unspent transaction for the exact amount required for the bet. To simplificate the transaction, more inputs nor change outputs are included.

The first output does not transfer any money, but includes: the list of oracles asked to participate; the channel to use for retrieving the full bet and the hash to compare the retrieved bet against. When an oracle sees itself in the list of oracles, it goes to the channel and retrieves the bet, compares it against the expected hash. Read the description and decides to participate or not.

The second and third outputs of this transaction moves most of the money into a joined account, so from now on it can only be moved by both players together. This represents the commitment to start the bet, as each one lock its own money under the other's will. In the case one of the players disappear, after τ_{Bet} the money can recovered by its owner.

The outputs 3 to 3 + (n - 1) are a small portion of the money, used for the oracle enrollments. This money does not includes a timeout because it is an small amount, if players are willing to pay the extra fee required to add this timeout, it can be added. This money is spend at the oracle enrollment as commitment from the players to the oracles.

This transaction spends $\mathcal{P} + \frac{c}{2}$ from each player: $n \cdot \frac{c}{n}$ goes to pay for oracle enrollments; $\mathcal{F}_{BetPromise}$ goes to pay this transaction's fees; and $2 \cdot \mathcal{P} - \mathcal{F}_{BetPromise}$ for paying the bet and its associated costs.

Oracle Enrollment

Oracles invited to participate needs to retrieve the Bet description as instructed in the *Bet Promise* transaction and decide whether to participate or not. When one or more oracles do not participate, players decide how to select new one(s), a waiting list is recommended to be selected in the first part of the protocol. When a oracle decides to participate, it builds and sends to the players its incription transaction, spending money from the *Bet Promise* transaction:

$T_{OracleEnrollment}$ (**in:** $T_{Oracleprevioustx}$,$T_{BetPromise}$)	
Inputs:	
$T_{Oracleprevioustx}[\alpha]$	σ
$T_{BetPromise}$[3 + Oracle Index]	$\mathcal{S}_A \wedge \mathcal{S}_B$
Outputs:	
([Registration] + c/n + [Oracle Payment] - $\mathcal{F}_{OracleEnrollment}$)	$(\mathcal{S}_{Oracle} \wedge \mathcal{S}_A \wedge \mathcal{S}_B)$ $\vee$ $(\mathcal{S}_{Oracle} \wedge \tau_{Bet})$
([Two Answers Penalty])	$((\mathcal{S}_A \vee \mathcal{S}_B) \wedge (AWins_o \wedge BWins_o))$ $\vee$ $(\mathcal{S}_{Oracle} \wedge \tau_{Two})$

Table 3.4: Oracle Enrollment

This transaction is the oracle's commitment with the bet and the players acceptance of it as oracle. The oracle sends [Registration] + [Oracle Payment] as initial payment (input 0), and players c/n from the *Bet Promise* (input 1). Oracle Index goes from 0 to n - 1.

The first output takes the money under the oracle and both players control, to be used in the *Bet* transaction. If the Bet transaction does not go into the blockchain, the oracle can claim this money after τ_{Bet}.

The second one is a penalty to be charge if the oracle misbehaves. If the oracle reveals that player A and player B wins, this money can be taken by any player. Otherwise the money can be claimed back by the oracle after τ_{Two}. Two Answers Penalty is the charge for voting twice, this must not be a small amount, as voting twice is the oracle's full responsibility. At this output the hash of the answers is revealed $(H(AWins_o), H(BWins_o))$, the oracle must keeps$AWins_o$ and $BWins_o$ secret, revealing this value is equivalent to choose a winner.

Bet

Once the required oracles are enrolled to participate, the "Bet" transaction is built by the players. The first two inputs contains what is remaining from the original players contributions to the prize, controlled by the players' joint account. Then, next n inputs are the outputs in the *Oracle Enrollment*, containing the oracles registration. This outputs are controlled by each oracle and both players.

Because inputs come from outputs controlled by all the participants, the transaction must be signed by both players and all the oracles:

T_{Bet} (**in:** $T_{BetPromise}$,$T_{OraclesEnrollment}$)	
Inputs:	
$T_{BetPromise}[0]$	$\mathcal{S}_A \wedge \mathcal{S}_B$
$T_{BetPromise}[1]$	$\mathcal{S}_A \wedge \mathcal{S}_B$
$T_{OracleEnrollment}[0]$	$\mathcal{S}_A \wedge \mathcal{S}_B \wedge \mathcal{S}_O$
$T_{...}[\dots]$	$\dots$ ‹n-1› $\dots$
Outputs:	
($\mathcal{P} + n/2$·([Registration] - [Oracle Payment] - $\mathcal{F}_{OracleEnrollment}$) - 1/2 ($\mathcal{F}_{BetPromise} + \mathcal{F}_{Bet}$))	$(\mathcal{S}_A \wedge (AWins_{\tilde{o}} \wedge \dots))$ $\vee$ $(\mathcal{S}_B \wedge (BWins_{\tilde{o}} \wedge \dots))$ $\vee$ $(\mathcal{S}_A \wedge \tau_{Two})$
($\mathcal{P} + n/2$·([Registration] - [Oracle Payment] - $\mathcal{F}_{OracleEnrollment}$) - 1/2 ($\mathcal{F}_{BetPromise} + \mathcal{F}_{Bet}$))	$(\mathcal{S}_A \wedge (AWins_{\tilde{o}} \wedge \dots))$ $\vee$ $(\mathcal{S}_B \wedge (BWins_{\tilde{o}} \wedge \dots))$ $\vee$ $(\mathcal{S}_B \wedge \tau_{Two})$
([Oracle Payment])	$(\mathcal{S}_o \wedge (AWins_o \vee BWins_o) \wedge \tau_{Bet})$ $\vee$ $\mathcal{S}_A \wedge \mathcal{S}_B \wedge \tau_{Reply}$
$(\dots)$	$\dots$ ‹n-1› $\dots$
([Oracle Payment])	$(AWins_o \wedge (BWins_{\hat{o}} \wedge \dots) \wedge \mathcal{S}_B)$ $\vee$ $(BWins_o \wedge (AWins_{\hat{o}} \wedge \dots) \wedge \mathcal{S}_A)$ $\vee$ $\mathcal{S}_o \wedge \tau_{Undue}$
$(\dots)$	$\dots$ ‹n-1› $\dots$

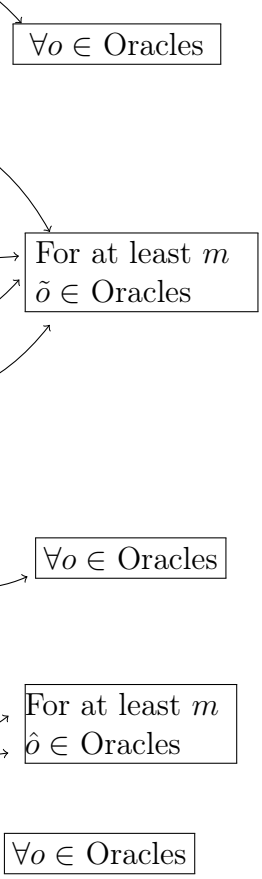

Table 3.5: Bet

The first two outputs are the prize, it can be spend by any of the two players' signature plus at least m votes of the oracles for that player. When any of the players sees m votes from the oracles it can get its prize. If the threshold is not met, half of this money goes to each player and we say the bet is not resolved, as there is no winner.

The next n outputs are the oracles's payment for answering. They can not be spend before τ_{Bet}, the moment when the event happen. It requires the vote of the oracle ($AWins_o$ or $BWins_o$) plus the oracle's signature. This output binds the vote with the oracle's payment, as they are required to make its vote public in order to get the payment. If the oracle does not answer after τ_{Reply} players are allowed to take this money back and the oracle can not further claim its payment.

The last n outputs are a withholding for the same amount of the oracle payment, if the oracle gives a wrong answer for the outcome, this money goes to the real winner. This way we take the payment out from the oracle, as it didn't give the right answer. If it behaves as expected, the oracle can spend this money some time after (T_{undue}) the bet is resolved.

There are four timeouts expressed in the transaction, table 3.6 describe each of them in chronological order, from sooner to later in the protocol execution.

Symbol	**Description**
τ_{Bet}	First timeout, this is the moment the event being used to decide the bet takes place. From this moment on oracles can vote for a winner and take its place.
τ_{Reply}	This timeout signals the time for the oracles to answer. After this timeout pass, players can take the oracle's payment if the oracle has not replied.
τ_{Undue}	If an oracle gave the wrong answer, players have until this timeout to take its payment back. After this timeout the oracle that behaved correctly can take the payment deposit back.
τ_{Two}	The last timeout, this could be the same than τ_{Undue}. Until this moment, players can take the two answers penalty for any oracle that made public the votes for both players. After this timeout, the oracle can take its deposit back.

Table 3.6: Timeouts

Timeouts are enforced using Relative Lock Time (RLT) as defined by the Bitcoin Improvement Proposal 68 (BIP 68). Their granularity is 512 seconds, roughly the same time it takes to produce a new block. As it is a 16 bits number, it's limited to be a year in the future, relative to the time its transaction was submitted to the blockchain.

Goes this next paragraph here? If we step out a little bit, the proposed protocol uses paid oracles to get a binary answer about an event outside the blockchain. This is not useful only for betting on that outcome. The oracles are[3] insensitive to the use given to their answer, they get paid anyway. Further applications of the protocol can be generalized from our proposal. Resolution of contractual disputes is an interesting topic, where parties agree to use arbitrator(s) to decide a missunderstanding. In this case oracles takes part in the protocol more like judge than a oracle.

[3] is insensitive the word?

3.5. Cost analysis

Most of the bibliography read simplifies the fee transactions to be 0, as this was the the cost of it for a long time. However as bitcoin use has increased, transactions are not free anymore[4], that's why we kept the fees in the explanation. So we can do the analysis with all the costs, not only the oracle payment.

In Bitcoin, fees are charged by byte, therefore bigger transactions will pay more money as fee. No matter how much money they spend.

At the moment we write this work, transactions paying 120 satoshi[5] usually gets into the next mined block. That is not too bad for money transferences, but as we have some big transaction we will do calculations with a fee of 15 per byte. This usually will not get our transactions into the first block, but into the first 15, this means it can take up to 3 hours to get the transactions in the blockchain. This is enough for an average case with enough time between timeouts. Players in a tight schedule can expend more money in fees and submit transactions faster.

It's impossible to give an exact value for the size of each transaction, as addresses and signatures do not have a fixed size, for the analysis we use average the size values, the fluctuation is under 5%. Table 3.7 has the size for the transactions used in the protocol, sizes are in bytes and we calculate the total using 7 as the number of oracles:

Transaction	**Constant size**	**Per oracle size**	**Total size**	**Fee [satoshi]**
Oracle Registration	239	0	239	3585
Bet Promise	1267	65	1722	25,830
Oracle Enrollment	776	0	776	11,640
Bet	617	445	3732	55,980
Player redeem prize	511	150	1561	23,415
Oracle redeem payment	355	0	355	5325
Oracle redeem undue	283	62	717	10,755
Oracle redeem two answers	323	0	323	4845
Player redeem wrong answer	338	70	828	12,420
Player redeem two answers	373	0	373	5595
Player redeem oracle doesn't answer	439	0	439	6585

Table 3.7: Transactions size and fee.

The first four transactions are the ones detailed in the previous section, however this transactions are not enough to make the complete cost analysis of a protocol execution. Redeem transactions are required to get the money into personal accounts, so we append

[4]We can still send transactions with no fees to the blochain, but it might take forever for them to get into it. As miner will prioritize transactions with fee to collect.

[5]A satoshi is the smallest unit of bitcoin on the blockchain. It is a one hundred millionth of a single bitcoin ($1 \cdot 10^{-8}$).

them.

For a successful run of the protocol, where there is a winner and every oracle behaves properly. Table 3.8 summarize the costs incurred by the players.

Item	**Cost [satoshi]**
Bet Promise fees	-25,830
Oracle Enrollments fee	-81,480
Bet transaction fee	-55,980
Oracles payment	-7 · [Oracle Payment]
Oracle first transfer	-c
Player redeem prize	-23,415
Total	-7 · [Oracle Payment] - c - 186,705

Table 3.8: Successful run, costs for the players

The 7 is the number of oracles, as set above. The value of c and [Oracle Payment] are parameters decided by the players, below we propose values for this constanst and discuss about the tradeoff on setting this values.

Table 3.9 summarize the cost and earning for an oracle that behaves correctly in a protocol execution.

Item	**Cost [satoshi]**
Initial deposit	-[Registration] - [Oracle Payment] - [Two Answers Penalty]
Payment	[Oracle Payment]
Redeem payment fee	-5325
Undue deposit	[Oracle Payment]
Undue redeem fee	-10,755
Two answers deposit	[Two Answers Penalty]
Two answers redeem fee	-5595
Total	[Oracle Payment] - [Registration] - 21,675

Table 3.9: Successful run, oracle cost and earning

The c constant is a transfer made from the players to the *Oracle Enrollment* bet, its objective is to regulate the oracle risk when accepting to participate in the bet. If an oracle send the *Oracle Enrollment* and the bet does not take place. The oracle will get back all its deposit + c/n - $\mathcal{F}_{OracleEnrollment}$.

Players might decide, in order to incentive oracles to participate, that there is no risk of losing money in this situation for oracles. So they set c such that $c/n > \mathcal{F}_{OracleEnrollment}$. Nonetheless, players might decide to make $c = 0$ because this simplifies the transaction and saves money in fees.

The [Registration] constant is an option to require the oracle to lock more money while participating in the bet. Making this constant big could help to have more committed oracles, as they have more money in the execution. Players have the option to set it ot zero, this will not save any fees. Probably the most important constant in this analysis is the [Oracle

Payment], it's the most expensive cost for the players and the way oracles earn money by participating.

As the tables show there is no dependency between the protocol costs and the amount of the bet. The cost for paying oracles is directly proportional to the number of them, and how much is each one getting paid. And the transaction cost depend on the number of oracles, the size of the transactions and the fee per byte.

So how much cost a run of the protocol, at the moment this work was written bitcoin was being exhange by USD 3800. If we replace this number in the first table we get the cost of the protocol for the players:

`7 * [Oracle Payment] + c + USD 7.095`

As the *Oracle Registration* transaction fees correspond to 3585 satoshis, we set the c/n parameter to be 5000, a little bit more than the cost of the fees. This gives a value of $c = 35,000$, so the cost of c is USD 1.33.

Argue about the value for the oracle payment is less technical, and probably every person reading this work will think in a different optimal value. We have simplified this payment to be equal for every oracle, but players might decide some oracles are more expensives than other ones. Oracles with more history or reputation might be worthy of a bigger payment. The amount being wagered is also an important factor to determine this number, if there is a really big amount of money at stake we might be willing to spend more money on this item, as this can decrease the chance of the oracles taking a bribe.

For this calculation we have defined a [Oracle Payment] of 200,000 satoshis, or USD 7.60. Simplifying the [Registration] to 0, running the protocol cost the users USD 61.62. In order to participate in the bet, each oracle needs to put a deposit of USD 30.40, and if behaves properly at the end of the bet will earn USD 6.40

To summarize, if player are willing to bet USD 1000, they will require to start with USD 1030.81, the winner would have win USD 969.19 at the end, and the loser would have spent 1030.81.

3.6. Communication

When the protocol is explained many things are get by granted, we just say oracles and players exchange data between them with not further discussion. When implementing this protocol the communication between participants can not be ignored, in this section we discuss a proposed model of communication for the protocol.

In Bitcoin communication in the protocol is not encrypted neither authenticated, the Bitcoin's protocol has incentives and cryptographic protection that make this possible. The protocol relies heavily on the Blockchain, so we take advantage of all these features and no communication requieres encryption nor authentication.

In order to start the protocol, players are required to know two things about the other. The other player's Bitcoin address and a communication method. A communication method besides the blockchain is required in order to discuss and get to an agreement in the bet parameters. This communication is required to be bidirectional.

The first step is to chat and decide the bet, if this communication is not secured, the *Bet Promise* transaction works as authentication method. Because at signing it, each player proves it control the private key corresponding to its address. And commits to the data the transaction holds.

When an oracle sees its id in the blockchain it needs to get the full bet description to decide whether to participate or not, as the full description hash is in the *Bet Promise* transaction, it can be retrieved using an insecure channel. If the oracle decides to participate it need to send the *Oracle Enrollment* transaction to the players, this transaction is signed with the oracle's private key, players must check it matches the address they selected. Oracle also sends the transaction script, as a P2SH only contains the hash of it, players are to check the transaction is as the protocol requires.

After all the oracles sent their transactions, players have all they need to create, sign and send the Bet transaction to the blockchain. No further communication off the blockchain is required from now on if every oracle behaves correctly. However, if a penalty is to be charged to an oracle Because it did not answer on time for example. communication off the blockchain is required to build and sign the transaction charging the penalty to the oracle.

3.6.1. Secured communication

As stated above, there is no need for secured communication in the protocol, however it might be desired for some participants, here we propose how to start a secure communication with no extra previous knowledge using a peer to peer channel.

Both players known each other's Bitcoin address, so the first step is to reveal to the other the address' public key using an insecure channel. This is verifiable by the receiver, as the address is the hash of the public key. Then they generate and send a random string in the insecure channel. After this, using the public key of the other player, a secure channel id and credentials to use it and the received random string are encrypted and sent using the insecure channel. This message is decrypted and a equality check is done, the random string received must match the one sent at the begining, this prevent replay attacks.

At this point both players know the secure channel and have the credentials to connect to it, secure communication can be started between them. This step is also reproducible with the oracles, nothing changes as the oracle's address is also known.

3.6.2. Channel

For test purposes we used a plain TCP connection between peers, and also a secure channel was implemented using CurveZMQ[6]. An authentication and encryption protocol on top of TCP that uses elliptic curves cryptography to protect the packages sent in the wire.

Establishing a peer to peer connection as channel might reveal more information than participants might want. A third party communication channel can be used to obscure our id from the other players, but it might be known by the third party controlling the channel. Several options can be used to obscure real id when communicating, using private navigation as the one provided by the Thor Network[7] can help when using a third party channel. Other approach is to use inherently anonymous communication protocols, as the one Orisi uses, BitMessage [33]. Which solution to use will vary from case to case and how difficult to track the participant wants to be.

3.7. Implementation

Bitcoin is widely used as value storage, most of the applications and library supporting it provide just a wallet and blockchain inspection functionality. Creating custom transactions is not a common functionality in bitcoin clients, and by the time this work started there was no library available to generate custom transactions.

In order to prove the viability of the proposed protocol and generate the transactions in the appendix 6.1 an implementation was written.

We used Java to implement all the protocol functionality. We divide our implementation in three logical modules to explain the work done:

1. **Bitcoin communication:** The official Bitcoin client exposes a RPC server to interact with the blockchain. This module provides an interface to the Bitcoin client, it allows our implementation to get information from the Blockchain and verify the validity of the transactions we generate.
2. **Bitcoin core:** This module expose Bitcoin objects with our required functionality. It is capable to generate custom transaction and sign them, it also provides transaction parsers and human readable views of them. Likely all this functionality would be in a "Bitcoin Library" if bitcoin were usually used to do smart contracts.
3. **Protocol implementation:** In this module we included the protocol specific functionality, it is able to generate and understand the Bets and the protocol transactions. It also exposes a distributed coin tossing implementation to select oracles randomly from a list.

Our implementation has a little bit more than 10 thousand lines of code. It includes:

[6] http://curvezmq.org/
[7] https://www.torproject.org/

- An encrypted and authenticated communication channel, which can start from an insecure channel only knowing the Bitcoin address of the other participant.
- A blockchain scan to compile a list of oracles from its Registration transactions.
- A distributed coin tossing implementation to select the oracles to use from the compiled list.
- A direct method to generate each of the required transactions in the protocol, including all the redeem transactions required to spend them.
- A check to verify the transactions received from the other player are valid and include the expected inputs and outputs.

We used this implementation and the official bitcoin client to prove the validity of the transactions generated. We want to be sure the transactions generated are valid Bitcoin transactions, so testing them using the official client is a crucial step.

Due to lack of libraries doing custom transactions, this implementation is a contribution not only for this protocol, but potentially for other applications using custom transactions.

Chapter 4

Discussion

4.1. Incentives

The ulterior motive for participating in the protocol is to win money, so we assume each player is driven by this. All of their action aims to this goal, win money. The protocol is designed with this assumption in mind and in the following paragraphs we discuss how the monetary incentives align participants to have a proper behaviour.

4.1.1. Players

The first thing to considerate is that players are paying fees for every transaction they send to the blockchain, in equal proportion. So, it is impossible the protocol can be aborted and both players get its money back. Once the first transaction is placed, at least one player will not recover its money: If the protocol does not finishes with a resolution, both will lose some money; If there is a winner he will get earnings and the other will lose everything.

One option to maximize the money won is to control the oracles. The first phase of the protocol and the second's player enforcement minimize the chances of selecting them. However there is another way to control the oracles, payments can be made to influence into the oracle's answer. Bribing the oracles is discused into the subsection 4.1.2.

Making the other player to lose money could be a motivation to some players, even if does not mean an earning for themself. As payments on timeouts and fees are equally distributed in the transactions, aborting the protocol at some point will mean an equal lose of funds for both players. If a player is willing to make the other one lose an amount of money, it will cost him the same amount. Other possible motivation could be to deprive the other player of its funds. But again for the same reason mentioned above, this will mean the player performing the attack should lock the same amount of money for the same period of time. A player can impair monetarily the other one, but is not for free, it will cost the same amount of money to do it. It can make the other player lose money, but only small amounts to be used as fees.

When it comes to funds depravation, the amount can be all the money involved, but it will not be lost, just locked for the time the bet defines as timeout.

4.1.2. Oracles

As the players, oracles also try to maximize their earnings, within the protocol that means to give the correct answer and collect its payment. But, there is an option to increase the earnings outside the protocol. Receiving money from a player to change their vote can give the oracle more money than answering correctly, as the incentive to change its vote can be bigger than the payment. A modified version of the bet transaction can used to do these payment, the player willing to pay the bribe can set the output to be spent with the answer he expects. So, in order to get the bribe, the oracle must reveal the answer the player pay to get. This make the problem even bigger, as no trust is required between a player and the oracle in order to cheat and change the answer.

This problem comes from the fact there is not source of truth accessible in the blockchain, in fact this is the problem the protocol tries to solve. The payment for answering correctly goes for the oracles that answer as the majority of them, not the ones that answer the truth, simply because that is the protocol truth. Bribing a majority of the oracles gives the oracles the bribe and also the payment. And this is the only useful bribe for the player, to change a minority of the answers does not gives him any benefit.

A way to mitigate this problem is to give the oracles certain reputation based on past behavior. This gives the oracles the incentive of behave properly in order to get long term earnings, as accepting bribes will erode their chances to be selected as oracles again. Players must consider this incentive when choosing oracles, the use of oracles with some kind of reputation decreases the chance of them taking a bribe.

4.2. Costs

At section 3.5 the cost of a successful transaction is provided, it sums up to over USD 60. A smaller number of oracles or less payment could bring down this number, however even after this the number will not be negligible. This has some important consequences for this protocol, its expensive to run, and unsuitable for small amount bets. Players might be willing to pay to run the protocol, but unlikely for small bets.

4.2.1. Individual Attacks

Oracle

Within the protocol, oracles maximize they earnings by participating and voting for winner. However, if oracles stop participating after they send the *Oracle Enrollment* and

$c/n > \mathcal{F}_{OracleEnrollment}$ they can also win money:

Item	Cost[USD]
Initial deposit	-[Oracle Payment] - [Two Answers Penalty]
Redeem initial deposit	[Oracle Payment] + c/n - $\mathcal{F}_{OracleEnrollment}$
Redeem initial deposit fee	- $\mathcal{F}_{RedeemInitialDeposit}$
Redeem two answers penalty	[Two Answers Penalty]
Redeem two answers penalty fee	- $\mathcal{F}_{RedeemTwoAnswersPenalty}$
Total	c/n - $\mathcal{F}_{OracleEnrollment}$ - $\mathcal{F}_{RedeemTwoAnswersPenalty}$

Table 4.1: Oracle exits after Oracle Enrollment

If c/n is bigger than the fees required to redeem the *Oracle Enrollment* transaction, oracle wins money by aborting after submitting this transaction.

For the players this bring other cost beside the c/n amount. They need to get a replacement for this oracle, a new smaller *Bet Promise* transaction can be sent. Which has an added cost, as it have to pay fees.

A malicious oracle can try to attack the players by aborting after the *Oracle Enrollment* transaction, resulting in an extra cost to the players. Decreasing the value of c makes this attack unlikely, as oracles will require to spend money in order to perform it. However decreasing this value might discourage oracles to participate in the bet. A second *Bet Promise* transaction to ask for another oracle is much cheaper than the original one, as it can link the bet data from to the first one, and only requires to list one oracle. So players can spend some extra money trying to get a new oracle, run the bet with less oracles or abort it. Running the bet with fewer oracles does is the only option that does not cost extra money.

This attack is limited, a malicious oracle can trigger the attack only once, as it will replaced or ignored after doing it. But if the same bet get multiple malicious oracles they can delay the bet long enough to forve player to abort it. This is extremely unlikely if oracles are being selected randomly from a big pool.

Player

If the bet faces a malicious player, it can abort the execution at any point before submitting the *Bet* transaction. As during the protocol both player contribute in the same way to pay it, if it gets cancelled at any moment, both players get to lose the same amount of money. So is feasible for a malicious player to attack the other one, but the cost of the attack is the same of the harm done. Other important measure the protocol has to limit the extent of this attack is to keep the bet money separated from the fees and oracle payments. A malicious player can not make the other one to lose more than the bet associated costs.

Colluded attacks

For every collusion involving the player and any number of oracles less than the majority threshold, the damage is limited for the bet associated costs, and they behave as the individual attacks described in sections 4.2.1 and 4.2.1. The cost of any of this attacks is the same of bigger than the damage done, so we say they are infeasible.

When one player and at least the majority threshold of the oracles are colluded, this player can take all the money to distribute it with the colluded oracles. This cost to the honest player all of the bet money plus the associated costs. This attack is discussed at section 4.1.2.

4.3. Previous Work

This protocol is not the first attempt to perform trustless gambling using crypto currencies. Is not even the first attempt to do it using Bitcoin in particular. In this section we try to differentiate this proposal with the existent solutions by highlighting their differences.

At section 2.5 we gave a short introduction to the existent solutions. Oraclize and Town Crier provide a secure interface to web sites. Making information of these websites available in the blockchain. A web site is a centralized data source, however combining multiple sites as data source give us a descentralized source. This could be a really good approach to do bets on massive events, as many websites already publish results of this events. If the bet is massive the cost of the oracles could be amortized among all the players.

However is difficult for this approach to work with non massive events, as web sites does not provide results for them. A second big difference with the proposed protocol is the platform they work on, Ethereum.

Winsome is another implementation on top of Ethereum that allows people to gamble against each other, but on random events. It emulates casino games as blackjack and roulette. As its target are random games, it's not suitable for real world event bets.

Ethereum unlike Bitcoin was not designed with the goal of being a currency, but a platform to write and enforce code execution in a distributed way. It's primary objective is to become a "Blockchain app platform" as they state in their website[1]. Because of this, their scripting language is turing complete and provides much more flexibility. This makes Ethereum the home for most of the blockchain applications. Recently, one of the Ethereum biggest app, the Decentralized Autonomous Organization (DAO), was subjected to an attack were attackers gained control of one third of the money it had. About 50 USD Million at the time of the attack. Most of the Ethereum community decided to "rollback" the history and give the DAO its money back. Immutability of the coin was therefore, broken.

Regardless bitcoin does not aim to be an app platform, people try to overcome its scripting

[1] https://ethereum.org/

limitations and write applications for it. Orisi is one of this applications that aims to solve the same problem we do. It works on top of bitcoin, one key difference is it's centralized oracle database. Other important difference is its secrecy, it uses BitMessage to keep the location and indentity of the participants secret. Although we could use BitMessage, our implementation uses direct TCP communication among the participants.

Probably the most deeper difference between our proposal and Orisi are the transactions used by each one. Orisi uses multisignature address to move the bet money into the control of the oracles and the players. This means, in order to redeem the priza,e the winner need to make a transaction and get it signed by a majority of the oracles. So oracles need to agree in the destination of the prize, otherwise they will not sign the transaction. Our proposal gets the oracles out from this decission and requires only the winner signature to spend the money. In order to claim its payment, oracles are required to reveal one of the two secrets they kept. When a majority of the oracles have revealed the secrets corresponding to the winner, this collect them all and claim its prize.

This has two main advantages: the first one, already mentioned, keeps the oracles out from deciding where does the prize goes. Oracles are not envolved in the transactions spending the prize; The second one is how it bounds the action of emiting the vote for the winner with getting paid. There is not pressure for the oracle in responding before the other do it, there is a fixed timeout defined from the beginning to vote. Oracles can be sure if they behave properly and vote on time they will be remunerated, even if they vote after a majority already did. In the multisignature address, the player could decide to wait only to the minimum required number of oracles and send the transaction to redeem the prize. Then the prize gets splitted in exactly the required majority. Oracles that answer on time after this are not guaranteed to get paid.

Chapter 5

Conclusion

Since the Bitcoin creation in 2008 more than a thousands of different cryptocurrencies have been created, most of them following its main design properties. Bitcoin is known and often criticized because its slow evolution and resistance to changes. This is accentuated for the overwhelming number of different and fast evolving currencies created.

A currency where rules does not change too much over time makes it a good store for value, and likely this Bitcoin's property contributes to make it the most valuable cryptocurrency, as measured by market capitalization. This is a desired feature for our algorithm, we do not want the value of the money put when the bet is placed to be too different of the one when the bet is resolved.

However, using Bitcoin did not come for free, this slow change rate keeps bitcoin script language very limited. When people talk about smart contract or scripting, Bitcoin is never the first currency coming to people's mind. The limited scripting, slow change rate and the biggest network made Bitcoin our choice: The script limitation poses a real challenge in the protocol presented, making this an open problem. As we discussed at section 4.3, the problem has many implemented solutions for currencies with more expressive scripting languages; The slow change rate helps to keep the currency value and the same rules from the start to the end of a bet, this makes the platform suitable for long term bets; Finally, having the largest network enforcing the protocol rules makes more difficult some known attacks, as Majority and Eclipse [19]. Where attackers can overtake the control of the network or part of it. Other cost of using a limited scripting language is the amount of instructions and transactions required to express the protocol, this is traduced to a higher fee cost.

The cost of running our proposed algorithm is prohibitively expensive for many cases. But we think there is people willing to pay the cost when doing bet on large sums of money, as the cost of the protocol would represent a small fraction of the bet. As there is no increase in the cost with the amount of money involved.

Most of the transaction required by the protcol are called "Non standard" in Bitcoin. Meaning even they are valid transactions and everyone recognizes them as that. Most of the miners will not mine them, so likely submitting the transactions will take more time than a

standard one. We hope new proposals that makes use of this available features, like this one, can move the Bitcoin community to embrace this "non estandard" transactions.

One of the most challenging problems to resolve for this protocol was to translate the protocol restrictions to Bitcoin transactions. Studied existing solutions are composed just by one or two different transactions. In Ethereum one smart contract is enough for the applications we discussed at section 2.5. For Orisi, only one transaction is post in the blockchain with the bet, all the other communication and set up required happens outside the blockchain. Our protocol uses the blochain widely, not only for money transfers, but also for oracle discovery and initial communication. This seeks to avoid extra dependecies and keep most of the protocol expressed in the same framework. This come to a cost, more and larger transactions. Which are not free anymore, for each byte we post in the blockchain within a transaction we are charged by the miner. This cost is discussed at section 3.5

A good example of this complexity is the transaction dumped at section 6.1.6. Bitcoin provides a multi-sig check function, which checks at least threshold signatures of the provided public keys are provided. At the Bet transaction we need to implement our own version of this language primitive, to get a multi-hash check function, which checks at least threshold preimages are provided for the written hash. This transaction is hard to read as it can not be written using loops (because the language does not support them), and even not used branches needs to be included to match the original hash in the Bet transaction output.

The initial decision of implementing the protocol prove to be very helpful when investigating limitations of the protocol. It allowed us to calculate the size of the required transactions, which were used to get a very accurate cost of the fees per transaction. Other limitation found thanks the implementation was the maximum size of an element on the Bitcoin stack: 520 bytes. Due this limitation our original implementation was able to use up to 6 oracles, after some script optimizations our implementatio support up to 7 oracles. With other optimizations we might be able to support one or two more oracles. However, without a change in the way Pay To Script Hash transactions work, or the hard limit of 520 bytes per element in the stack. It is impossible for our protocol to support a bigger number of oracles.

Many of the work on the implementation were basic tools to manipulate and generate custom transactions. This code is a contribution not only for our protocol, but also for people willing to use the bitcoin with custom transactions, as we do.

Gambling is a really big and heavily regulated industry, Winsomeio, oraclizeit and this protocol are tools to make this industry more flexible. Allowing people to fully control its money while gambling, a difference from the regular casinos or bet sites. As cryptocurrencies allows people to stay away from a centralized government issuer, this kind of tools allows people to steo out from centralized gambling sites. There is yet another big difference, people running protocols on top of blockchains can see every line of code executed and how decissions are taken, making gambling a transparent process, opossed as it is today. Where people uses only the public interface of a web site or physical casino. Never seeing what is really going on "behind the scenes".

We believe this work is a contribution for both worlds, it introduces new options on the gambling industry. And also adds a new application on top of Bitcoin, using it as more

than a value store currency. Bitcoin introduces a method to store and transfer value without relying in a central authority as previous currencies did before. The presented protocol uses this property and introduces an option to do these transfers subject to real world events, while keeping the lack of central control to achieve it. Therefore the objective for this work is accomplished.

While working on this we realized bets are only one example of external people or events deciding destination of somebody else's money. We could think in our oracles as "judges" in a dispute. And instead of providing an event outcome into the blockchain, they can decide on any subject, acting as arbiter on a controversy.

Small adjustements would be required to the protocol, as removing the random selection, probably judges will be agreed beforehand by the players. Other required change is to remove the penalization by answering different from the majority. It is perfectly fine and even desirable to have different opinions among the judges, and the ones with the minority should not be punished.

It might be also desirable to have a larger number of oracles participating in a bet. However, this requires a change in the Bitcoin implementaion, which is not under our control.

Future work might also include sharing oracles among different Bets. This could help on making the protocol cheaper but adds new issues, as paying the oracle. Our proposal pays the oracles in the same Bet transaction, get all the users to finance the oracles would require a new way to do it.

Other interesting change would be to decrease the transactions number or transactions' size to make the fees more affordables. One of the last additions to Bitcoin was "Segregated Witness", it solves transaction malleability, which opens the door to the use of off the blockchain transactions. This is an option to decrease the number of required transactions posted in the blockchain, and therefore decrease the amount of fees paid.

Bibliography

[1] Ross Anderson, Charalampos Manifavas, and Chris Sutherland. Netcard—a practical electronic-cash system. In *International Workshop on Security Protocols*, pages 49–57. Springer, 1996.

[2] Marcin Andrychowicz, Stefan Dziembowski, Daniel Malinowski, and Lukasz Mazurek. Secure multiparty computations on bitcoin. In *Security and Privacy (SP), 2014 IEEE Symposium on*, pages 443–458. IEEE, 2014.

[3] Europen Central Bank. Virtual Currency Schemes. 2012. URL: `https://www.ecb.europa.eu/pub/pdf/other/virtualcurrencyschemes201210en.pdf` (visited on 01/03/2017).

[4] Simon Barber, Xavier Boyen, Elaine Shi, and Ersin Uzun. Bitter to better—how to make bitcoin a better currency. In *International Conference on Financial Cryptography and Data Security*, pages 399–414. Springer, 2012.

[5] Manuel Blum. Coin flipping by telephone a protocol for solving impossible problems. *ACM SIGACT News*, 15(1):23–27, 1983.

[6] Jean-Paul Boly, Antoon Bosselaers, Ronald Cramer, Rolf Michelsen, Stig Mjølsnes, Frank Muller, Torben Pedersen, Birgit Pfitzmann, Peter De Rooij, Berry Schoenmakers, et al. The esprit project cafe—high security digital payment systems. *Computer Security—ESORICS 94*:217–230, 1994.

[7] David Chaum. Achieving electronic privacy. *Scientific american*, 267(2):96–101, 1992.

[8] David Chaum. Blind signatures for untraceable payments. In *Advances in cryptology*, pages 199–203. Springer, 1983.

[9] David Chaum, Amos Fiat, and Moni Naor. Untraceable electronic cash. In *Proceedings on Advances in cryptology*, pages 319–327. Springer-Verlag New York, Inc., 1990.

[10] Victor Costan and Srinivas Devadas. Intel sgx explained. *IACR Cryptology ePrint Archive*, 2016:86, 2016.

[11] Christian Decker and Roger Wattenhofer. Information propagation in the bitcoin network. In *Peer-to-Peer Computing (P2P), 2013 IEEE Thirteenth International Conference on*, pages 1–10. IEEE, 2013.

[12] Whitfield Diffie and Martin Hellman. New directions in cryptography. *IEEE transactions on Information Theory*, 22(6):644–654, 1976.

[13] Cynthia Dwork and Moni Naor. Pricing via processing or combatting junk mail. In *Annual International Cryptology Conference*, pages 139–147. Springer, 1992.

[14] Liam Edwards-Playne. The invention of the blockchain. 2013. URL: `https://medium.com/@liamzebedee/the-invention-of-the-blockchain-fe25be0caebc` (visited on 05/24/2017).

[15] Michael J Fischer. The consensus problem in unreliable distributed systems (a brief survey). In *International Conference on Fundamentals of Computation Theory*, pages 127–140. Springer, 1983.

[16] Sally M Gainsbury, Alex Russell, Robert Wood, Nerilee Hing, and Alex Blaszczynski. How risky is internet gambling? a comparison of subgroups of internet gamblers based on problem gambling status. *New Media & Society*, 17(6):861–879, 2015.

[17] Mark Griffiths and Andrew Barnes. Internet gambling: an online empirical study among student gamblers. *International Journal of Mental Health and Addiction*, 6(2):194–204, 2008.

[18] MD Griffiths. Crime and gambling: a brief overview of gambling fraud on the internet. *Internet journal of criminology*, 2010.

[19] Ethan Heilman, Alison Kendler, and Aviv Zohar. Eclipse attacks on bitcoin's peer-to-peer network. In.

[20] Sarah Jane Hughes, Stephen T Middlebrook, and Broox W Peterson. Developments in the law concerning stored-value cards and other electronic payments products, 63 bus. *Law*, 237:237–40, 2007.

[21] Hrishikesh Huilgolkar. Introducing winsome.io. 2017. URL: `https://blog.winsome.io/introducing-winsome-io-86ba703f33c` (visited on 05/24/2017).

[22] John Ferlito John Ferlito Robert Lord. Oraclize docs. 2016. URL: `http://docs.oraclize.it/` (visited on 07/09/2017).

[23] Tomasz Kolinko and the Orisi team. Orisi white paper. 2014. URL: `https://github.com/orisi/wiki/wiki/Orisi-White-Paper` (visited on 06/07/2017).

[24] Anna Lysyanskaya and Zulfikar Ramzan. Group blind digital signatures: a scalable solution to electronic cash. In *Financial cryptography*, pages 184–197. Springer, 1998.

[25] John L McMullan and Aunshul Rege. Online crime and internet gambling. *Journal of Gambling Issues*:54–85, 2010.

[26] Andrew Miller and Joseph J LaViola Jr. Anonymous byzantine consensus from moderately-hard puzzles: a model for bitcoin, 2014.

[27] Satoshi Nakamoto. Bitcoin: a peer-to-peer electronic cash system, 2008.

[28] John Naughton. Is blockchain the most important it invention or our age? 2016. URL: `https://www.theguardian.com/commentisfree/2016/jan/24/blockchain-bitcoin-technology-most-important-tech-invention-of-our-age-sir-mark-walport` (visited on 05/24/2017).

[29] Tatsuaki Okamoto and Kazuo Ohta. Universal electronic cash. In *Annual International Cryptology Conference*, pages 324–337. Springer, 1991.

[30] Bart Preneel. *Analysis and design of cryptographic hash functions*. PhD thesis, Katholieke Universiteit te Leuven, 1993.

[31] Bhiru Shelat and Florian N Egger. What makes people trust online gambling sites? In *CHI'02 Extended Abstracts on Human Factors in Computing Systems*, pages 852–853. ACM, 2002.

[32] Felix Sun and Peitong Duan. Solving byzantine problems in synchronized systems using bitcoin, 2014.

[33] Jonathan Warren. Bitmessage: a peer-to-peer message authentication and delivery system. *white paper (27 November 2012), https://bitmessage. org/bitmessage. pdf*, 2012.

[34] Fan Zhang, Ethan Cecchetti, Kyle Croman, Ari Juels, and Elaine Shi. Town crier: an authenticated data feed for smart contracts. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 270–282. ACM, 2016.

Chapter 6

Appendix

6.1. Transactions

Examples of each one of the most important transactions in the protocol, parsed and explained. Transactions are represented using a json-like format with the field name and its value. In the blockchain, the trasaction is a concatenation of all the values exposed here, the field names are added to make the transaction human readable. These transactions are generated for the testnet of Bitcoin, this blockchain works as test for changes to be introduced in Bitcoin. It uses the same bitcoin client and rules, the main difference is their value, "bitcoins" in the testnet do not have a monetary value, so they are easier to access to.

6.1.1. Oracle Registration

```
{
  version: 1
  inputs counter: 1
  inputs: {
    {
      prev_tx_hash: 25FCA8C2AA9EC0FBFE418EF18EFF7E683B13EA344F77B5
                    ↪ 3AF48C9D50F980BD45
      prev_idx: 0
      script_length: 107
      script: 483045022100C2CDC0F9C2297D0359F41E6D878AF962966B7DA2
              ↪ 1126BA530C4117B47786165902201F1A667A2433F3336CD8
              ↪ DB220D954585EE3DEECB333DBAABA1E438BD344857D10121
              ↪ 0293CE4CCDC6B49EE87DF835EC2AD77D92822A52444C425D
              ↪ 03C7EDFAD09FCC2E52
      sequence_no: 4294967295
    }
  }
```

```
  outputs_counter: 2
  outputs: {
    {
      value: 0
      script_length: 38
      script: [OP_RETURN, OP_PUSH_36_bytes,
               49276D20616E206F7261636C652120526561647920746F2070
                 ↪ 726F766964652064617461]
    }
    {
      value: 20968080
      script_length: 25
      script: [OP_DUP, OP_HASH160, OP_PUSH_20_bytes,
               360973EAE34A60DD1EAB3D18705B9EF1C531B57F,
               OP_EQUALVERIFY, OP_CHECKSIG]
    }
  }
  lock_time: 0
}
```

The input spends money in the oracle's control, this is outside the scope of the protocol, so we do not parse it.

The first output is a protocol dependent constant, it uses the OP_RETURN opcode to push the data mark the output as invalid. In this example the constant is the string "I'm an oracle! Ready to provide data". This constant is used to identify an oracle enrollment in the blockchain, so players can find and use the oracles.

The second output is the change. In this case the protocol does not require to spend any money in this transaction, so the oracle takes all the money back to an address it controls.

6.1.2. Bet Promise

```
{
  version: 1
  inputs counter: 7
  inputs: {
    {
      prev_tx_hash: 294B8B68DEC454C90A25C8CCA68EEBCF2280205AC17781
                    ↪ 39527DD64317E6CE14
      prev_idx: 0
      script_length: 106
      script: 47304402204C295245BD9389FC6DD7405F12DC04F6E8FB0E999A
              ↪ 08ABE8C14CCEFF7D6414E002207B6F5C77210B0CF2DC67DD
              ↪ 7E90A8526CC130DA28A87323B612FB030C417447C9012102
              ↪ A52B895664A1D2738B3C1FDEDC112F9B2F1FEF0800150E46
              ↪ 83CF3344AEEE149E
```

```
  sequence_no: 4294967295
}
{
  prev_tx_hash: A5C7E5CB12835D4B4CC8CBBEF7F6C1D6761F1C779DE396
                ↪ 179CFAF85A8B966434
  prev_idx: 0
  script_length: 106
  script: 473044022044166548 2EFADF4FE4BA306888233D2D9825722A1A
          ↪ 3FBF192D8BA84A5771C86B02201EB740CEF6ABB362DB9AA5
          ↪ ADB6E57E5248B220B0558165D1B371411B651F4666012102
          ↪ 23522078AF81C1A6D028F37B3446ABD29412EFBDF08EAFB1
          ↪ 122B223340E32D1E
  sequence_no: 4294967295
}
{
  prev_tx_hash: F66078FCFED4721EB83D7E329E443CA647F78C9FFF6A77
                ↪ C0DD9362932A4BB1D8
  prev_idx: 0
  script_length: 107
  script: 483045022100997F4732B6AFFC2BD95342DCE884ACA906A4A48B
          ↪ 1A841A2CDE07C8C2CD87ADCB0220356B28388D35826AA010
          ↪ 18C154DBA2F1E916F786C3F3E6BD68DB339AEDCDAFAA0121
          ↪ 036E623812F3F4042FA0B981483CB1562E2FE3654E90CB52
          ↪ DF5938932E3B616DEC
  sequence_no: 4294967295
}
{
  prev_tx_hash: 322881B20E1C25A7BC7A4FF0947128F05353BDC1186016
                ↪ 907F70316A117B9F34
  prev_idx: 1
  script_length: 106
  script: 47304402205AFE3147A8CCE8A7950BE4B3CE0D65C0EBFA1D39FF1
          ↪ 354CD28C7243F7B0CFE5B022072F165E7DE88CD8A7D8C3B4E
          ↪ B8899900CCB9AEE7CC4E4D3FC0D206EC0094252E012102E99
          ↪ 3E533772F9851A962C59D1819D5641D48B45DC7F6B2D2E2CF
          ↪ 3C34369F7DB5
  sequence_no: 4294967295
}
{
  prev_tx_hash: FBC0BA746FECF73D12DEC825BC92630C1C014551F3818D
                ↪ 0CEAE06B71999A644E
  prev_idx: 0
  script_length: 106
  script: 473044022036DE6627456A7DE119093F1E949705D090F00F521E
          ↪ 169C023E8BA9325B44175D022002A2A66272116DDAA378FE
          ↪ 89A0C0E0D6F202CCA9B7CF3CC75E2C01EC19ADE4D9012102
          ↪ E993E533772F9851A962C59D1819D5641D48B45DC7F6B2D2
```

```
            ↪ E2CF3C34369F7DB5
    sequence_no: 4294967295
  }
  {
    prev_tx_hash: 741ADE91531D0537B94BBAB93E438D487E2EB4AF070DA2
                  ↪ 10888527D4710FA780
    prev_idx: 1
    script_length: 106
    script: 473044022070E8DAFBC001E12A6D2477391D493B0E22A7C1F044
            ↪ 23E47CC517E9FDA83CE0E802207F6A16960C13FAEA4386BB
            ↪ 2405478B4D82B39911CB64DEDD5C36F363A10434A5012103
            ↪ 4D00A2B430B6999974A29A168B56A12C0065A6CF95181DB3
            ↪ E209CD5B14332568
    sequence_no: 4294967295
  }
  {
    prev_tx_hash: 4004CC1568D1041710924139C1206A89CCCA5C8D2D072C
                  ↪ 426E9E75C94B31F7EF
    prev_idx: 0
    script_length: 107
    script: 483045022100FCED70DE0C66BC3E3B10F6F1A3AC6849BE625AA6
            ↪ 0D28E1F5A7E36E0FFCB372C6022054CC3D92DD4914FEFA9F
            ↪ 2BC8DA77182C4691E754794741C5DD7A1C44080ED29F0121
            ↪ 02E993E533772F9851A962C59D1819D5641D48B45DC7F6B2
            ↪ D2E2CF3C34369F7DB5
    sequence_no: 4294967295
  }
}
outputs_counter: 10
outputs: {
  {
    value: 0
    script_length: 257
    script: [OP_RETURN, OP_PUSH_254_bytes,
            17426574206C6F6F6B696E6720666F72206F7261636C657305
              ↪ 226D6B536737437248366E7A584145753753454E6F6B7
              ↪ 371636E4252625663477838682 26D6E41764644697434
              ↪ 50745A5338507334474D73456F7167436E3543624E473
              ↪ 36B4B226D69687976764D756A47317A57416946515239
              ↪ 3850336A67614C4A4C557A504E6172226D6D334551536
              ↪ E74344A506659505063586A64616F61666A5062445235
              ↪ 6231444C57226D726567504274733375 6D6B4B386F7A4
              ↪ 55764486E35324E516D34384D51566D597394AC3992DD
              ↪ 3B975DCA40927169D471EBE00F19E601206C6F63616C6
              ↪ 86F73743A343332342C203137322E31392E322E35343A
              ↪ 38383736]
  }
```

```
{
  value: 99892970
  script_length: 23
  script: [OP_HASH160, OP_PUSH_20_bytes,
           029EF2FC0D7C46BDCC4F07E517F6CC99F3711A66, OP_EQUAL]
}
{
  value: 99892970
  script_length: 23
  script: [OP_HASH160, OP_PUSH_20_bytes,
           003F90F5C12455CABD2F31685048ED4EDB3457AA, OP_EQUAL]
}
{
  value: 60000
  script_length: 23
  script: [OP_HASH160, OP_PUSH_20_bytes,
           B0A5676BE17E30499FFF05011D1D404770C48A40, OP_EQUAL]
}
{
  value: 60000
  script_length: 23
  script: [OP_HASH160, OP_PUSH_20_bytes,
           B0A5676BE17E30499FFF05011D1D404770C48A40, OP_EQUAL]
}
{
  value: 60000
  script_length: 23
  script: [OP_HASH160, OP_PUSH_20_bytes,
           B0A5676BE17E30499FFF05011D1D404770C48A40, OP_EQUAL]
}
{
  value: 60000
  script_length: 23
  script: [OP_HASH160, OP_PUSH_20_bytes,
           B0A5676BE17E30499FFF05011D1D404770C48A40, OP_EQUAL]
}
{
  value: 60000
  script_length: 23
  script: [OP_HASH160, OP_PUSH_20_bytes,
           B0A5676BE17E30499FFF05011D1D404770C48A40, OP_EQUAL]
}
{
  value: 220550000
  script_length: 25
  script: [OP_DUP, OP_HASH160, OP_PUSH_20_bytes,
           66E2E39AB1C5E98A8180DE876CC4743D3292E14A, OP_EQUALVERIFY,
```

```
                OP_CHECKSIG]
    }
    {
      value: 531650000
      script_length: 25
      script: [OP_DUP, OP_HASH160, OP_PUSH_20_bytes,
                73F245C95CAF31F4449B8259D6CEB248A57262F8, OP_EQUALVERIFY,
                OP_CHECKSIG]
    }
  }
  lock_time: 0
}
```

The inputs is all the money required to run this bet, it comes from both oracles previously owned addresses.

The first output pushes: A constant to identify the data, in this case the string "Bet looking for oracles"; The bet description's hash; The channel to type and address to stablish communication with the players. In this case an IP address and a port.

The second and third outputs move most of the money to a joint account, to be used in the Bet transaction.

The fourth to the eighth outputs set apart the money to be used in the oracle enrollment.

Ninth and tenth are the change for both players, de difference from what they got from its input and the required to run the bet.

Second to eighth outputs use Pay To Script Hash. This means we do not get o see the real script required to spend them, just the hash of it. The script (and the data required to evaluate it) must be provided when spending this outputs.

6.1.3. Oracle Enrollment

```
{
  version: 1
  inputs counter: 4
  inputs: {
    {
      prev_tx_hash: 25FCA8C2AA9EC0FBFE418EF18EFF7E683B13EA344F77B5
                    ↪ 3AF48C9D50F980BD45
      prev_idx: 0
      script_length: 107
      script: [OP_PUSH_72_bytes
                3045022100D0F089625E7E1198D48D6DF2AACF0A5AAB42853C
                  ↪ A48EE0B458DCC91D7438FB0502200DFA820300B8389C6
                  ↪ ED256387F78A3C2B30453B504520DB96F47DA57349883
                  ↪ [ALL]
```

```
          OP_PUSH_33_bytes
          0293CE4CCDC6B49EE87DF835EC2AD77D92822A52444C425D03
            ↪ C7EDFAD09FCC2E52]
  sequence_no: 4294967295
}
{
  prev_tx_hash: F3378F87464223D458156E561D147B1E8C50DD55860A69
                ↪ 065F7B3BD1D96F8EAB
  prev_idx: 0
  script_length: 106
  script: [OP_PUSH_71_bytes
          304402206F48E436F5DDC97A74D7CF96BBD3BEF6252964F90B1
            ↪ 6F0F22E67CB93FD7D18610220441F11AB77AC454258808
            ↪ BD641183ADD71D6B8EB7DD455A9F400D2315958DF[ALL]
          OP_PUSH_33_bytes
          0293CE4CCDC6B49EE87DF835EC2AD77D92822A52444C425D03C
            ↪ 7EDFAD09FCC2E52]
  sequence_no: 4294967295
}
{
  prev_tx_hash: F65ED60FAF8C89F16E89939B1EBD5AC91FCCDA1EE4EBE1
                ↪ 1BD4DA203469C714CA
  prev_idx: 0
  script_length: 107
  script: [OP_PUSH_72_bytes
          3045022100D32C220D13956398B7C1EBB7858FD4272F40ACD5
            ↪ BDB76E1D2FAF3A2E709047C8022065CC1C9AD993483B78
            ↪ C8394DBAB2CDEC1C67B49576BF0CCEB8B1DEFAB52254
            ↪ [ALL]
          OP_PUSH_33_bytes
          0293CE4CCDC6B49EE87DF835EC2AD77D92822A52444C425D03
            ↪ C7EDFAD09FCC2E52]
  sequence_no: 4294967295
}
{
  prev_tx_hash: B7C9A9B3870295EF98367502E599210E1046410C9FA154
                ↪ 7321E8E026CAEB5590
  prev_idx: 3
  script_length: 218
  script: [OP_0 OP_PUSH_72_bytes
          3045022100E3D14D93F42FB897D3DAB079122843CAC1DBE72A
            ↪ 22132689001FB452BCA144330220573015E3783DD47A42
            ↪ F49CB5A77E85CCA87049AEF4775B942DD740AF09962F
            ↪ [ALL]
          OP_PUSH_71_bytes
          304402203DE0D1CD2DE543795F79933DACCADFAAFF8D5D631E
            ↪ FF039EAC25CC8D759E696902207F48C2244049A8BF8764
```

```
                ↪ B74C7659C704E48E89E095968DBE4F76D17F23E1EA
                ↪ [ALL]
              OP_PUSH_71_bytes
              {
                [OP_2 OP_PUSH_33_bytes
                  02EF5471E4BB8B2B6709C700057DBA30F2F9D418ACD2C
                  ↪ D156B997F865034C257D1
                  OP_PUSH_33_bytes
                  034D00A2B430B6999974A29A168B56A12C0065A6CF9518
                  ↪ 1DB3E209CD5B14332568
                  OP_2 OP_CHECKMULTISIG]
              }
            ]
    sequence_no: 4294967295
  }
 }
 outputs_counter: 3
 outputs: {
  {
    value: 3001780
    script_length: 23
    script: [OP_HASH160, OP_PUSH_20_bytes,
             1DBB37095E8EC83A0C37D4AF62778AA3035C8B7E, OP_EQUAL]
  }
  {
    value: 10000000
    script_length: 23
    script: [OP_HASH160, OP_PUSH_20_bytes,
             A17855C80692945036967371A75ACD35003DEDDC, OP_EQUAL]
  }
  {
    value: 232075000
    script_length: 25
    script: [OP_DUP, OP_HASH160, OP_PUSH_20_bytes,
             360973EAE34A60DD1EAB3D18705B9EF1C531B57F,
             OP_EQUALVERIFY, OP_CHECKSIG]
  }
 }
 lock_time: 0
}
```

First three inputs belong to the oracle, this money come from transactions outside the protocol. And we can see the input of this transactions is the required data for a Pay to Public Key Hash, the public key and its corresponding signature.

The fourth input is spending the fourth (index 3) output from the Bet Promise transaction. It contains the output script requesting both players signatures using the OP_CHECKMULTISIG

opcode. It also contains the two required signatures.

The first output moves the money to a joint account controlled by both players and the oracle.

The second output is the two answers penalty deposit.

The third output is the oracle's change, it gives the non used money back to the oracle.

6.1.4. Bet

```
{
  version: 2
  inputs counter: 7
  inputs: {
    {
      prev_tx_hash: 2138072E6593FFBFECAB73D9F2B819E84274DAA78811DD
                    ↪ 54D994DF0972EBB657
      prev_idx: 1
      script_length: 227
      script: [OP_PUSH_71_bytes
               3044022069A810EBE1EEF38E7D4EC7606A5B98A2FAC417B3A1
                ↪ 7C012EA3249316AF5A6FD4022067F532EAE45500B6B0FA
                ↪ C8C42FE53E312406706FA2CEF47F11DD4567D0A517
                ↪ [ALL]
               OP_1 OP_PUSH_71_bytes
               30440220160EC827ECE0FD655F17D3C157C9BFAFF36ADF1CEB
                ↪ 23640E47B67D02E0B26A7802205ADE0BE3833FBD6E2B8A
                ↪ C37A4991A7F6954F1962E2CEDA86A824B9C65025BB
                ↪ [ALL]
               OP_PUSH_80_bytes
               {
                 [OP_PUSH_33_bytes
                  02EF5471E4BB8B2B6709C700057DBA30F2F9D418ACD2CD1
                   ↪ 56B997F865034C257D1
                  OP_CHECKSIGVERIFY
                  OP_IF
                    OP_PUSH_33_bytes
                    034D00A2B430B6999974A29A168B56A12C0065A6CF95
                     ↪ 181DB3E209CD5B14332568
                    OP_CHECKSIGVERIFY
                  OP_ELSE
                    OP_PUSH_3_bytes FA0140
                    OP_CHECKSEQUENCEVERIFY OP_DROP
                  OP_ENDIF
                  OP_1]
               }
```

```
        ]
  sequence_no: 18446744073709551615
}
{
  prev_tx_hash: 2138072E6593FFBFECAB73D9F2B819E84274DAA78811DD
                ↪ 54D994DF0972EBB657
  prev_idx: 2
  script_length: 227
  script: [OP_PUSH_71_bytes
           304402202C6CFD8F24F24C093F37A7C00C983E0530F44F038B
            ↪ CC0D88DCF8F1906681A2B8022065C689F5AF3C0B81F719
            ↪ 816A19228217E592588EFC0EA9B900BF57C8FA6FCF
            ↪ [ALL]
           OP_1 OP_PUSH_71_bytes
           304402202B23B5CF5B175225BCD571CB5384C1E0AE8D991771
            ↪ DFF56B3C65825C840D19DF02204677B925F2F0DC8AD860
            ↪ 22867A005FF2F61F4DB642D7A12FA3815FF5EAE701
            ↪ [ALL]
           OP_PUSH_80_bytes
           {
             [OP_PUSH_33_bytes
              034D00A2B430B6999974A29A168B56A12C0065A6CF95181
               ↪ DB3E209CD5B14332568
              OP_CHECKSIGVERIFY
              OP_IF
                OP_PUSH_33_bytes
                02EF5471E4BB8B2B6709C700057DBA30F2F9D418ACD2C
                 ↪ D156B997F865034C257D1
                OP_CHECKSIGVERIFY
              OP_ELSE
                OP_PUSH_3_bytes FA0140
                OP_CHECKSEQUENCEVERIFY
                OP_DROP
              OP_ENDIF
              OP_1]
            }
        ]
  sequence_no: 18446744073709551615
}
{
  prev_tx_hash: 9EF6EFAD5B2C9EF387D19D878DA83E9743FAAE0A235BF4
                ↪ 8B049D9C86CB84F84A
  prev_idx: 0
  script_length: 337
  script: [OP_0 OP_PUSH_71_bytes
           3044022025E3CB3C98561F4E96E63A992895015531CE029B64
            ↪ 2D4E64F3D0A6D606E4302502200A9A9CDC8C7B9D19C81D
```

```
            ↪ 8AEB51DEDE329FE710C5877E3B923BF8B58C2ED678
            ↪ [ALL]
          OP_PUSH_71_bytes
          304402203B778E24A9E64161925AC8C6D47937250E47AE251C
            ↪ B3D47FFB4EE01F0D6DC96202205DB43C82081391471D28
            ↪ 20B530D51CB27B90B187B22FC8F775F5E5CEAD7307
            ↪ [ALL]
          OP_1 OP_PUSH_72_bytes
          3045022100C8CD147FC265CF5A1A7AE63EE89070C308D1A54A
            ↪ 0885F6048556B76F1CFA0D95022039141CBDA7D289A467
            ↪ 8F296C4687AA46CAA9B1887CC9BF31FCB6705052F37E
            ↪ [ALL]
          OP_PUSH_116_bytes
          {
            [OP_PUSH_33_bytes
             03E4BAB917EA0BD852820237A59595CB3B06F9492EF6EC5
              ↪ BEC16F5A89B5717E1F2
             OP_CHECKSIGVERIFY
             OP_IF
               OP_2
               OP_PUSH_33_bytes
               02EF5471E4BB8B2B6709C700057DBA30F2F9D418ACD2C
                ↪ D156B997F865034C257D1
               OP_PUSH_33_bytes
               034D00A2B430B6999974A29A168B56A12C0065A6CF951
                ↪ 81DB3E209CD5B14332568
               OP_2
               OP_CHECKMULTISIGVERIFY
             OP_ELSE
               OP_PUSH_3_bytes FA0140
               OP_CHECKSEQUENCEVERIFY
               OP_DROP
             OP_ENDIF
             OP_1]
          }
        ]
  sequence_no: 18446744073709551615
}
{
  prev_tx_hash: FBBEC4193E0B7BB96208BB7CC5EA3834D324E422CDA367
                  ↪ E57C65511E9696D810
  prev_idx: 0
  script_length: 336
  script: [OP_0 OP_PUSH_71_bytes
           30440220266A8174C9A75DA1C8FC7A43276B8FF2FEA79F87A9
            ↪ B9D10141729E0FA685A4CD0220316F6FBE91934C236BD4
            ↪ 608BB80C67AEFADCFD67F5C177039A70BF5A292380
```

```
            ↪ [ALL]
          OP_PUSH_71_bytes
          304402204C6A3C1E25615C2BF7B33F5637C78C24A1E1E33880
            ↪ 6675DC2B1905AFBCC2A06302200929503 99A27279FA952
            ↪ 71ABE31C3F4D370853649665C69093016EDE6E9054
            ↪ [ALL]
          OP_1 OP_PUSH_71_bytes
          304402201501B998021EF8B77884EF66931B9B3B024021CAB0
            ↪ EE7FA98A0085D79EEF0180022016BBA25B58C5D66AECC0
            ↪ 431144B722528A587C1005EA4EBD83EA866DA95576
            ↪ [ALL]
          OP_PUSH_116_bytes
          {
            [OP_PUSH_33_bytes
             03C26AADD6B66332134182E8FCA8C238A7C24448D43C368
              ↪ 860BF1C7488F3CDC171
             OP_CHECKSIGVERIFY
             OP_IF
               OP_2
               OP_PUSH_33_bytes
               02EF5471E4BB8B2B6709C700057DBA30F2F9D418ACD2C
                ↪ D156B997F865034C257D1
               OP_PUSH_33_bytes
               034D00A2B430B6999974A29A168B56A12C0065A6CF951
                ↪ 81DB3E209CD5B14332568
               OP_2
               OP_CHECKMULTISIGVERIFY
             OP_ELSE
               OP_PUSH_3_bytes FA0140
               OP_CHECKSEQUENCEVERIFY
               OP_DROP
             OP_ENDIF
             OP_1]
          }
        ]
  sequence_no: 18446744073709551615
}
{
  prev_tx_hash: 3F32A95A1F811F2DF13192B1EB09C48C2C53491ADF7E9C
                  ↪ 9D5152FB64ECCE4758
  prev_idx: 0
  script_length: 336
  script: [OP_0 OP_PUSH_71_bytes
           3044022069269B2D9053B13EDEA05F8A2726ADDE9DB8EE36DC
            ↪ E20BE9F80A3C4FF02C3EC702202018BA6BACF254AF9183
            ↪ 426C668C2EE5FCBAF17A062EAF3EAFE7D529697FA8
            ↪ [ALL]
```

```
            OP_PUSH_71_bytes
            304402202B43221C29EF9A32048BC883BC91E8BEEBF03EB300
             ↪ BEDA73B2790364C496A06E02204321EAED90C7A06518C6
             ↪ E6FE497786A6ED4B07EA7D9EB22E5745B137E98C2C
            [ALL]
            OP_1 OP_PUSH_71_bytes
            30440220554BF72107A0F64007E0C36D38FBDE9C82FE6E4B70
             ↪ 56482FE7B23009A14AFC76022001BC85C822F49EBAC44E
             ↪ 37DF62414AD0A36D1DDAAE22B5911302BFD735FD49
            [ALL]
            OP_PUSH_116_bytes
            {
              [OP_PUSH_33_bytes
               034D3259B9F364B8642A7E5937BD1C678E73C8353B0D525
                ↪ FF4C3181ADDE1CCA95B
               OP_CHECKSIGVERIFY
               OP_IF
                 OP_2
                 OP_PUSH_33_bytes
                 02EF5471E4BB8B2B6709C700057DBA30F2F9D418ACD2C
                  ↪ D156B997F865034C257D1
                 OP_PUSH_33_bytes
                 034D00A2B430B6999974A29A168B56A12C0065A6CF951
                  ↪ 81DB3E209CD5B14332568
                 OP_2
                 OP_CHECKMULTISIGVERIFY
               OP_ELSE
                 OP_PUSH_3_bytes FA0140
                 OP_CHECKSEQUENCEVERIFY
                 OP_DROP
               OP_ENDIF
               OP_1]
            }
          ]
  sequence_no: 18446744073709551615
}
{
  prev_tx_hash: 9DBCE4D777148100FA1D7E11B0F6FA568413601B24948F
                 ↪ 598D67036BF9BADB3B
  prev_idx: 0
  script_length: 337
  script: [OP_0 OP_PUSH_71_bytes
           3044022000DBD27D25C4BF0E1E1E2493DB3309AEC5F181288D
            ↪ 1B6925463C93B7A06E1F730220293A8F9652A374E59E9D
            ↪ 09CC129DD5753D83F811FE42E6B4096F4E6FCA1DDC
            ↪ [ALL]
           OP_PUSH_71_bytes
```

```
            3044022016BA7E45A2D6A8174DE2A0476FA7A0E40C853CE793
             ↪ F71BDB5839FA824C253E1902206AEF00B3DD3EA8DCEBB1
             ↪ BF4B6741D28254A9384F11F72560BB63EC4E80570E
             ↪ [ALL]
            OP_1 OP_PUSH_72_bytes
            30450221009BD014D23E09693DE13CBE20CABCE8D341D22792
             ↪ 0D2884AC43584CC4770CBE3A022008A4D78E48620B56B5
             ↪ 148AC4CC214D962127DE10DD4977F447698D0FEE8C1C
             ↪ [ALL]
            OP_PUSH_116_bytes
            {
              [OP_PUSH_33_bytesi
               03840940CAB26409E8239C463655A4668108FA7C3B5F2A4
                ↪ F7FAFFD2C4D00047247
               OP_CHECKSIGVERIFY
               OP_IF
                 OP_2
                 OP_PUSH_33_bytes
                 02EF5471E4BB8B2B6709C700057DBA30F2F9D418ACD2C
                  ↪ D156B997F865034C257D1
                 OP_PUSH_33_bytes
                 034D00A2B430B6999974A29A168B56A12C0065A6CF951
                  ↪ 81DB3E209CD5B14332568
                 OP_2 OP_CHECKMULTISIGVERIFY
               OP_ELSE
                 OP_PUSH_3_bytes FA0140
                 OP_CHECKSEQUENCEVERIFY
                 OP_DROP
               OP_ENDIF
               OP_1]
            }
          ]
  sequence_no: 18446744073709551615
}
{
  prev_tx_hash: 66AFA218E07ABE0D8242666FF170522F1CCED81006A1A3
                 ↪ F9CF0E10808749F89E
  prev_idx: 0
  script_length: 338
  script: [OP_0 OP_PUSH_71_bytes
           304402204CE28022808810B4DF1FEC09F5D14CB7369BCAB67D
            ↪ 320C23F5CEE468484B1CF102202C2D4CBCB7E917D49160
            ↪ 57DAA5B19F807C28FDBA7BD487FC9E87C2894F4EB2
            ↪ [ALL]
           OP_PUSH_72_bytes
           3045022100D624642E9EE4FD9E30ACA11579BD86B4646E846E
            ↪ 2E7619E82C24D2E602A2F82F0220054972431AB9F49FA7
```

```
                ↪ 1D345DD2C638ED49CCBE540A4F42F5E354293DDF5E4F
                ↪ [ALL]
              OP_1 OP_PUSH_72_bytes
              3045022100CE64EBBCC4D1125A59A5A989C69777051E840685
                ↪ A4094D03411867192E072B1C02200D1E53F93EA142968C
                ↪ ABDE879C599F465EBA818094FF4D867A7BEC6C13B048
                ↪ [ALL]
              OP_PUSH_116_bytes
              {
                [OP_PUSH_33_bytes
                 0320E9F9ED3209E2A2552354FDD632245B13A0019B067EE
                  ↪ 4537ECFD9C6CCC17582
                 OP_CHECKSIGVERIFY
                 OP_IF
                   OP_2
                   OP_PUSH_33_bytes
                   02EF5471E4BB8B2B6709C700057DBA30F2F9D418ACD2C
                    ↪ D156B997F865034C257D1
                   OP_PUSH_33_bytes
                   034D00A2B430B6999974A29A168B56A12C0065A6CF95181DB3E209CD5B14332568
                   OP_2
                   OP_CHECKMULTISIGVERIFY
                 OP_ELSE
                   OP_PUSH_3_bytes FA0140
                   OP_CHECKSEQUENCEVERIFY
                   OP_DROP
                 OP_ENDIF
                 OP_1]
              }
            ]
    sequence_no: 18446744073709551615
  }
}
outputs_counter: 12
outputs: {
  {
    value: 92281150
    script_length: 23
    script: [OP_HASH160
             OP_PUSH_20_bytes E5A86D80F148897C3813C3981E9940CF328AAC49
             OP_EQUAL]
  }
  {
    value: 92281150
    script_length: 23
    script: [OP_HASH160
             OP_PUSH_20_bytes 2DCCF04275EB961DD4ED9E87EBC9B948C88E70E1
```

```
          OP_EQUAL]
}
{
  value: 3000000
  script_length: 23
  script: [OP_HASH160
          OP_PUSH_20_bytes B8ED98E1F64E5A6278093054F201B1704B5D4519
          OP_EQUAL]
}
{
  value: 3000000
  script_length: 23
  script: [OP_HASH160
           OP_PUSH_20_bytes D48B76D17895B4B8D6364F60FE3E0D734F3D8958
           OP_EQUAL]
}
{
  value: 3000000
  script_length: 23
  script: [OP_HASH160
           OP_PUSH_20_bytes E84ABC5DAEBF11FFEC60E479ED8C17651837008D
           OP_EQUAL]
}
{
  value: 3000000
  script_length: 23
  script: [OP_HASH160
           OP_PUSH_20_bytes 56D2E9575A80B96BFA3B9B1435A91DEFCBE77781
           OP_EQUAL]
}
{
  value: 3000000
  script_length: 23
  script: [OP_HASH160
           OP_PUSH_20_bytes 7C4DCB244DA6DF5D472A7F88BA2C4D21867E99F7
           OP_EQUAL]
}
{
  value: 3000000
  script_length: 23
  script: [OP_HASH160
           OP_PUSH_20_bytes CD5FA2DEF69CA02FB959FA79663C7D9366B1BAE1
           OP_EQUAL]
}
{
  value: 3000000
  script_length: 23
```

```
      script: [OP_HASH160
               OP_PUSH_20_bytes 632C59C13A5D3CE501CD772242D1E1278B0CE456
               OP_EQUAL]
    }
    {
      value: 3000000
      script_length: 23
      script: [OP_HASH160
               OP_PUSH_20_bytes 9D2E8D88757AA4227B72EA2111201909305B9DFA
               OP_EQUAL]
    }
    {
      value: 3000000
      script_length: 23
      script: [OP_HASH160
               OP_PUSH_20_bytes 639C1223300F80B287AF33819512EF861D7027E5
               OP_EQUAL]
    }
    {
      value: 3000000
      script_length: 23
      script: [OP_HASH160
               OP_PUSH_20_bytes 8EA5E31EF1F503BAFB9CEF5B3C02F03C0DD945A9
               OP_EQUAL]
    }
  }
  lock_time: 0
}
```

First two inputs spend the the second and third output at the Bet Promise transaction. They include each player's signature and the output script. If we look closely, the output script is symmetric, as seen in table 3.3. We can see the main difference is the position of each player's public key. Other interesting data to notice is the "FA0140" constant that appears, it is the τ_{Bet}[1].

Next five inputs are the money coming from each oracle's enrollment. At this inputs we can see three signatures are used, both players and the corresponding oracle. We can see two of the public keys are repeated in the previous inputs, however the signature provided for them are not equal. This is because the way Bitcoin specify transactions are signed: When signing a Pay To Script Hash all others input's scripts are set emptied. And the one being sign is set to the output's script the input spends. This is how Bitcoin ensures signatures are tied to its input.

6.1.5. Oracle Answer

[1]The interpretation of this bytes can be found at Bitcoin Improvement Proposal 0112. https://github.com/bitcoin/bips/blob/master/bip-0112.mediawiki

```
{
  version: 2
  inputs counter: 1
  inputs: {
    {
      prev_tx_hash: 9CAA0B7F15D18DAC9034DB114452ACD405C53B4BB1C20C
                    ↪ CCE0A7C6ADC0CF226A
      prev_idx: 2
      script_length: 269
      script: [OP_PUSH_22_bytes
               1AE3DB90D7B53CB9C1173900C356AE2D27CFB11B68EB
               OP_1 OP_PUSH_71_bytes
               30440220737C8D9FBEC9E236CC29A4A39D8C9AE157B4EC591E
                ↪ 09B92A1093CD0C5D495E4E02201FF11CB6D5EFDFE9CA9D
                ↪ 16BA4FD35623C634F0E7942D97B5D9EDB411B3B0E4
                ↪ [ALL]
               OP_1
               OP_PUSH_170_bytes
               {
                 [OP_IF
                    OP_PUSH_33_bytes
                     03E4BAB917EA0BD852820237A59595CB3B06F9492EF
                     ↪ 6EC5BEC16F5A89B5717E1F2
                     OP_CHECKSIGVERIFY
                     OP_PUSH_3_bytes FA0140
                     OP_CHECKSEQUENCEVERIFY
                     OP_DROP
                     OP_IF
                       OP_HASH160
                       OP_PUSH_20_bytes
                       50918176ACCC76AFA983FBC196F72A13169E3706
                       OP_EQUALVERIFY
                     OP_ELSE
                       OP_HASH160
                       OP_PUSH_20_bytes
                       CB2349648A50A14AF307D60946B36299D323C58B
                       OP_EQUALVERIFY
                     OP_ENDIF
                   OP_ELSE
                     OP_PUSH_33_bytes
                     02EF5471E4BB8B2B6709C700057DBA30F2F9D418ACD2
                      ↪ CD156B997F865034C257D1
                     OP_CHECKSIGVERIFY
                     OP_PUSH_33_bytes
                     034D00A2B430B6999974A29A168B56A12C0065A6CF95
                      ↪ 181DB3E209CD5B14332568
                     OP_CHECKSIGVERIFY
```

```
                    OP_PUSH_3_bytes 320240
                    OP_CHECKSEQUENCEVERIFY
                    OP_DROP
                  OP_ENDIF
                  OP_1]
              }
            ]
      sequence_no: 4194810
    }
  }
  outputs_counter: 1
  outputs: {
    {
      value: 2950160
      script_length: 25
      script: [OP_DUP, OP_HASH160
               OP_PUSH_20_bytes
               360973EAE34A60DD1EAB3D18705B9EF1C531B57F
               OP_EQUALVERIFY
               OP_CHECKSIG]
    }
  }
  lock_time: 0
}
```

The first input of this transaction is giving a vote for winner to the first player. The value pushed is the 22-byte secret which hash is expected by the output script at the nested if-else block. We can see there are two options in this block, they only differ in the hash expected. They represent the vote for each player, and both have the same validity for this script. The second branch for the output script (not used by this transaction) requires both players' signatures and to be submitted after the timeout "320240", corresponding to τ_{Reply}. At posting this transaction the oracle makes public its vote, after the required number of oracles does this, the winner can take these hash and redeem its prize.

The output of this transaction is a simple Pay To Public Key Hash, where the oracles has decided to move its payment.

6.1.6. Winner Prize Collect

```
{
  version: 2
  inputs counter: 2
  inputs: {
    {
      prev_tx_hash: 17640D0C824C575BE3F393491B32D1B585E5CA28F36BDB
                    ↪ 2D2F98EFB3006B4535
      prev_idx: 0
```

```
script_length: 567
script: [OP_PUSH_71_bytes
         304402207DC931CE42CCBD552FFCD75D57AF6A5E49B204ACAA
          ↪ 55AAB145C9181E44BA6B4D02207A031F9AF4EF605163A6
          ↪ C364660E6F7F06E49968823FF76679AF46CB1CFA5D
          ↪ [ALL]
         OP_PUSH_33_bytes
         02EF5471E4BB8B2B6709C700057DBA30F2F9D418ACD2CD156B
          ↪ 997F865034C257D1
         OP_PUSH_1_bytes 51
         OP_PUSH_19_bytes
         3AFFA9D769B5807CE4F9764771A0F966CE02A6
         OP_PUSH_23_bytes
         5FDDD56DB498CFB29C514B3A27A6F685FF40449E9974CA
         OP_PUSH_21_bytes
         5A0A3C2DB3CB1DE19C0896FED80B4335FB6C503194
         OP_1
         OP_1
         OP_PUSH_388_bytes
         {
           [OP_IF
             OP_IF
               OP_0
               OP_TOALTSTACK
               OP_HASH160
               OP_DUP
               OP_PUSH_20_bytes
               010E7DAFABDD0E8EED77B2C1F715F3071F710D3E
               OP_EQUAL
               OP_IF
                 OP_DROP
                 OP_HASH160
               OP_ELSE
                 OP_FROMALTSTACK
                 OP_1ADD
                 OP_TOALTSTACK
               OP_ENDIF
               OP_DUP
               OP_PUSH_20_bytes
               4D2021B46ED7B29EFC20AFB4C771AAE341AEBE5D
               OP_EQUAL
               OP_IF
                 OP_DROP
                 OP_HASH160
               OP_ELSE
                 OP_FROMALTSTACK
                 OP_1ADD
```

```
    OP_TOALTSTACK
  OP_ENDIF
  OP_DUP
  OP_PUSH_20_bytes
  774594000150D5935E3B1F63187EFCEB1E36CB83
  OP_EQUAL
  OP_IF
    OP_DROP
    OP_HASH160
  OP_ELSE
    OP_FROMALTSTACK
    OP_1ADD
    OP_TOALTSTACK
  OP_ENDIF
  OP_DUP
  OP_PUSH_20_bytes
  590F74932E25BE9A958FD19D39BE52E2A4CB9D3F
  OP_EQUAL
  OP_IF
    OP_DROP
    OP_HASH160
  OP_ELSE
    OP_FROMALTSTACK
    OP_1ADD
    OP_TOALTSTACK
  OP_ENDIF
  OP_DUP
  OP_PUSH_20_bytes
  276C972344745AADA362D4CECA7A25301EE47D86
  OP_EQUAL
  OP_IF
    OP_DROP
    OP_HASH160
  OP_ELSE
    OP_FROMALTSTACK
    OP_1ADD
    OP_TOALTSTACK
  OP_ENDIF
  OP_DROP
  OP_FROMALTSTACK
  OP_2
  OP_LESSTHANOREQUAL
  OP_VERIFY
OP_ELSE
  OP_PUSH_3_bytes
  6A0240
  OP_CHECKSEQUENCEVERIFY
```

```
    OP_DROP
  OP_ENDIF
  OP_DUP
  OP_HASH160
  OP_PUSH_20_bytes
  66E2E39AB1C5E98A8180DE876CC4743D3292E14A
  OP_EQUALVERIFY
  OP_CHECKSIG
OP_ELSE
  OP_0
  OP_TOALTSTACK
  OP_HASH160
  OP_DUP
  OP_PUSH_20_bytes
  99B6793C6944C55AB1195957545746055FEA0593
  OP_EQUAL
  OP_IF
    OP_DROP
    OP_HASH160
  OP_ELSE
    OP_FROMALTSTACK
    OP_1ADD
    OP_TOALTSTACK
  OP_ENDIF
  OP_DUP
  OP_PUSH_20_bytes
  2A3FB2A0E3DF3219579F5AF968A7AE118C7287FE
  OP_EQUAL
  OP_IF
    OP_DROP
    OP_HASH160
  OP_ELSE
    OP_FROMALTSTACK
    OP_1ADD
    OP_TOALTSTACK
  OP_ENDIF
  OP_DUP
  OP_PUSH_20_bytes
  2C1771354527252791F83A78AC5CEBD7427ACB59
  OP_EQUAL
  OP_IF
    OP_DROP
    OP_HASH160
  OP_ELSE
    OP_FROMALTSTACK
    OP_1ADD
    OP_TOALTSTACK
```

```
                OP_ENDIF
                OP_DUP
                OP_PUSH_20_bytes
                A843DBB79C247128B7BD223A80CA3551201515A3
                OP_EQUAL
                OP_IF
                  OP_DROP
                  OP_HASH160
                OP_ELSE
                  OP_FROMALTSTACK
                  OP_1ADD
                  OP_TOALTSTACK
                OP_ENDIF
                OP_DUP
                OP_PUSH_20_bytes
                32FB95E3F35819519703A58B016F02C9C4943BFC
                OP_EQUAL
                OP_IF
                  OP_DROP
                  OP_HASH160
                OP_ELSE
                  OP_FROMALTSTACK
                  OP_1ADD
                  OP_TOALTSTACK
                OP_ENDIF
                OP_DROP
                OP_FROMALTSTACK
                OP_2
                OP_LESSTHANOREQUAL
                OP_VERIFY
                OP_DUP
                OP_HASH160
                OP_PUSH_20_bytes
                4C48CA1BE269DF0539BFD40D27AC797101B7FFC9
                OP_EQUALVERIFY
                OP_CHECKSIG
              OP_ENDIF]
           }
          ]
  sequence_no: 4194810
}
{
  prev_tx_hash: 17640D0C824C575BE3F393491B32D1B585E5CA28F36BDB
                ↪ 2D2F98EFB3006B4535
  prev_idx: 1
  script_length: 567
  script: [OP_PUSH_72_bytes
```

```
3045022100DDC7EE0D84A2B67A8EE5CF6328DE45776F518705
 ↪ 617C3E817FD368A74C6FC00E0220013496B5C3E7925887
 ↪ E32FB1253E2E1ECFC4A8D17AA6586BF84AF3F8A7EE97
 ↪ [ALL]
OP_PUSH_33_bytes
02EF5471E4BB8B2B6709C700057DBA30F2F9D418ACD2CD156B
 ↪ 997F865034C257D1
OP_PUSH_1_bytes 51
OP_PUSH_19_bytes
3AFFA9D769B5807CE4F9764771A0F966CE02A6
OP_PUSH_23_bytes
5FDDD56DB498CFB29C514B3A27A6F685FF40449E9974CA
OP_PUSH_21_bytes
5A0A3C2DB3CB1DE19C0896FED80B4335FB6C503194
OP_0
OP_PUSH_388_bytes
{
  [OP_IF
    OP_IF
      OP_0
      OP_TOALTSTACK
      OP_HASH160
      OP_DUP
      OP_PUSH_20_bytes
      99B6793C6944C55AB1195957545746055FEA0593
      OP_EQUAL
      OP_IF
        OP_DROP
        OP_HASH160
      OP_ELSE
        OP_FROMALTSTACK
        OP_1ADD
        OP_TOALTSTACK
      OP_ENDIF
      OP_DUP
      OP_PUSH_20_bytes
      2A3FB2A0E3DF3219579F5AF968A7AE118C7287FE
      OP_EQUAL
      OP_IF
        OP_DROP
        OP_HASH160
      OP_ELSE
        OP_FROMALTSTACK
        OP_1ADD
        OP_TOALTSTACK
      OP_ENDIF
      OP_DUP
```

```
  OP_PUSH_20_bytes
  2C1771354527252791F83A78AC5CEBD7427ACB59
  OP_EQUAL
  OP_IF
    OP_DROP
    OP_HASH160
  OP_ELSE
    OP_FROMALTSTACK
    OP_1ADD
    OP_TOALTSTACK
  OP_ENDIF
  OP_DUP
  OP_PUSH_20_bytes
  A843DBB79C247128B7BD223A80CA3551201515A3
  OP_EQUAL
  OP_IF
    OP_DROP
    OP_HASH160
  OP_ELSE
    OP_FROMALTSTACK
    OP_1ADD
    OP_TOALTSTACK
  OP_ENDIF
  OP_DUP
  OP_PUSH_20_bytes
  32FB95E3F35819519703A58B016F02C9C4943BFC
  OP_EQUAL
  OP_IF
    OP_DROP
    OP_HASH160
  OP_ELSE
    OP_FROMALTSTACK
    OP_1ADD
    OP_TOALTSTACK
  OP_ENDIF
  OP_DROP
  OP_FROMALTSTACK
  OP_2
  OP_LESSTHANOREQUAL
  OP_VERIFY
OP_ELSE
  OP_PUSH_3_bytes
  6A0240
  OP_CHECKSEQUENCEVERIFY
  OP_DROP
OP_ENDIF
OP_DUP
```

```
  OP_HASH160
  OP_PUSH_20_bytes
  4C48CA1BE269DF0539BFD40D27AC797101B7FFC9
  OP_EQUALVERIFY
  OP_CHECKSIG
OP_ELSE
  OP_0
  OP_TOALTSTACK
  OP_HASH160
  OP_DUP
  OP_PUSH_20_bytes
  010E7DAFABDD0E8EED77B2C1F715F3071F710D3E
  OP_EQUAL
  OP_IF
    OP_DROP
    OP_HASH160
  OP_ELSE
    OP_FROMALTSTACK
    OP_1ADD
    OP_TOALTSTACK
  OP_ENDIF
  OP_DUP
  OP_PUSH_20_bytes
  4D2021B46ED7B29EFC20AFB4C771AAE341AEBE5D
  OP_EQUAL
  OP_IF
    OP_DROP
    OP_HASH160
  OP_ELSE
    OP_FROMALTSTACK
    OP_1ADD
    OP_TOALTSTACK
  OP_ENDIF
  OP_DUP
  OP_PUSH_20_bytes
  774594000150D5935E3B1F63187EFCEB1E36CB83
  OP_EQUAL
  OP_IF
    OP_DROP
    OP_HASH160
  OP_ELSE
    OP_FROMALTSTACK
    OP_1ADD
    OP_TOALTSTACK
  OP_ENDIF
  OP_DUP
  OP_PUSH_20_bytes
```

```
                    590F74932E25BE9A958FD19D39BE52E2A4CB9D3F
                    OP_EQUAL
                    OP_IF
                      OP_DROP
                      OP_HASH160
                    OP_ELSE
                      OP_FROMALTSTACK
                      OP_1ADD
                      OP_TOALTSTACK
                    OP_ENDIF
                    OP_DUP
                    OP_PUSH_20_bytes
                    276C972344745AADA362D4CECA7A25301EE47D86
                    OP_EQUAL
                    OP_IF
                      OP_DROP
                      OP_HASH160
                    OP_ELSE
                      OP_FROMALTSTACK
                      OP_1ADD
                      OP_TOALTSTACK
                    OP_ENDIF
                    OP_DROP
                    OP_FROMALTSTACK
                    OP_2
                    OP_LESSTHANOREQUAL
                    OP_VERIFY
                    OP_DUP
                    OP_HASH160
                    OP_PUSH_20_bytes
                    66E2E39AB1C5E98A8180DE876CC4743D3292E14A
                    OP_EQUALVERIFY
                    OP_CHECKSIG
                  OP_ENDIF]
              }
]
      sequence_no: 4194810
    }
  }
  outputs_counter: 1
  outputs: {
    {
      value: 184385480
      script_length: 25
      script: [OP_DUP OP_HASH160 OP_PUSH_20_bytes
              66E2E39AB1C5E98A8180DE876CC4743D3292E14A
              OP_EQUALVERIFY OP_CHECKSIG]
```

```
    }
  }
  lock_time: 0
}
```

Inputs spend the first two outputs of the bet transaction. They contain the public key and the signature of the winner, and a majority of the votes for this player. In this case we have 5 oracles, and the threshold is set to 3, so inputs include three votes on its script, we can see these votes at the beginning of the script. The large and repetitive script is one of the costs of using a non turing complete language. Because there are no loops availables, we need to write everything explicitly, like an unrolled loop. This script contains three main branches, one for the timeout case, where a new constant is visible: "6A0240", which is the τ_{Two}. And one for each winner, we can see at both of them the 20-byte hash of the expected votes. Most of the size in this output are these hash, in this case 10 of them, 2 per each oracle participating. An interesting property of this transaction is all the signatures required are generated by the winner, third parties are not required directly to redeem the bet prize.

The script implements an operation similar to a multi-sig check, but instead of signatures it checks multiple hash. It works by testing the provided pre image candidates against the expected hash, by hashing and then testing equality. Each time a pre image does not match a hash, we increase a counter stored in the alt stack, at the end of the script we check this counter to be less than 2. The difference between the total number of oracles and the threshold. It requires ordering in the provided candidates, respecting the order hash are written.

The output is a Pay To Public Key Hash which moves the money to an account controlled by the player.