A Fair Protocol for Data Trading Based on Bitcoin Transactions

Sergi Delgado-Segura, Cristina Pérez-Solà,
Guillermo Navarro-Arribas, Jordi Herrera-Joancomartí
Department of Information Engineering and Communications,
Universitat Autònoma de Barcelona

Abstract

On-line commercial transactions involve an inherent mistrust between participant parties since, sometimes, no previous relation exists between them. Such mistrust may be a deadlock point in a trade transaction where the buyer does not want to perform the payment until the seller sends the good and the seller does not want to do so until the buyer pays for the purchase. In this paper we present a fair protocol for data trading where the commercial deal, in terms of delivering the data and performing the payment, is atomic since the seller cannot redeem the payment unless the buyer obtains the data and the buyer cannot obtain the data without performing the payment. The protocol is based on Bitcoin scripting language and the fairness of the protocol can be probabilistically enforced.

1 Introduction

From its first uses, computer networks have been applied as a business base ground to perform commercial transactions. Obviously, on-line interactions impose some restrictions on how such transactions can be performed since there is no physical contact between the parties. One of the first problems that on-line economic transactions faces is distrustful that parts in the transaction may have on each other. When a buyer is in a regular shop buying some groceries, whether the seller will provide the groceries in a bag first or the buyer will give the money for that purchase before is irrelevant since the on-site transaction reduces the distrustful of the parts. However, if the economic transaction is on-line such situation implies a disadvantage for the party that performs the fist step of the protocol. If the buyer pays before receiving the purchase, the seller could act maliciously by not sending the goods and, conversely, if the seller delivers the goods before getting paid, the buyer could disappear with the product without paying. Such situation is solved in real transactions by creating some trust relationship between buyer and seller. Such trust is somehow based on the fact that the buyer is willing to pay before the product is delivered because standard on-line payment systems, like credit cards or bank transfers, have the possibility to be reversed. So the buyer has some confidence that if something goes wrong, for instance, the product is not delivered, he could prove it, the system could reverse the payment and the he would not be at a disadvantage.

However, the situation is different when blockchain based cryptocurrencies are the payment system for purchases. Blockchain based cryptocurrencies avoid the double-spending problem of digital cash systems by maintaining a general ledger in which all transactions are stored. The main property of such ledger is its immutability which makes payments final once a transaction is deep enough in the blockchain. Once the payment is firmly included in the blockchain, it is impossible to reverse such payment, unless the payee of such transaction unilaterally agrees to return the money. Such mechanism leaves the buyer in a weaker position when the payment is performed before obtaining the goods.

The best approach to solving this unequal advantage of the parties in a purchase is to set the whole transaction atomic in the sense that the payment and good delivery takes place at the same time. With this approach, in case one of the parties does not complete his part, any exchange is finished (neither delivery nor payment). In fact, such situation is the best emulation of on-site shop scenario where the payment and the delivery take place at the same time, almost atomically. Of course, translating such approach to a virtual environment implies that goods traded would be restricted to digital data.

The contribution of this paper is the following. We propose a fair protocol for data trading. Our protocol is fair since none of the participants have an advantageous position in the execution of the protocol. The protocol is atomic in the sense that either it is fully executed, ending the buyer with the data and the seller with the payment, or no party incurs in any loss. Our proposal is a practical one, based on Bitcoin scripting language, and can be deployed using existing technology, in contrast to other theoretical approaches that are reviewed in Section 2.

The rest of the paper is organized as follows. In Section 2 we review the state of the art in fair exchange protocols. Section 3 provides some background on Bitcoin transactions, its scripting language and the main relevant transactions used in our protocol. In Section 4 the fair protocol for data trading is presented and its main properties analyzed. Finally, Section 5 concludes the paper.

2 Related work

Our proposal can be seen as a fair exchange protocol where two parties agree in the exchange of some data for a given value (in this case measured in Bitcoins). Usually, fair exchange protocols can be used to sign a contract between two parties stating such exchanges and the conditions under the exchange has to be carried. The signature is produced so that no party can gain advantage over the other. We rely in Bitcoin as a way to actually implement the exchange contract. The contract does not need to be enforced by other parties (even if the signature of the contract is fairly produced by both parties) because Bitcoin itself *executes* the contract in the form of a transaction script.

Fair exchange protocols are usually divided into two party protocols and protocols requiring a trusted third party (TTP). Two party protocols, provide a gradual exchange of messages or information between the two parties, to gradually decrease uncertainty and increase fairness in the transaction, without the need for a TTP. First proposed in [7], the idea is for the two parties to exchange secrets bit by bit allowing them to verify the correctness of the received bits. This idea was also proposed in other approaches [9] more specifically for the signature of contracts. Probabilistic protocols for fair exchange were introduced in [4], where the goal is for the parties to end up with a given probability on the fairness (commitment to the contract by the two parties) at a given time (or step).

Regarding the use of a TTP, we usually distinguish between online and offline TTP. The online TTP acts as an intermediary between the two parties ensuring the fairness of the exchange [20, 10]. On the other hand an offline TTP only acts in case of dispute and does not participate in the protocol if all parts act honestly, also called an *optimistic* fair exchange [2, 21, 3].

Authors refer to the notion of perfect fairness (also called strong fair-exchange) when a party cannot leave the protocol with a small advantage over the opponent [16]. Perfect fair exchange usually requires the use of TTP-based protocols, although there are several alternatives to implement it. Some of them relay in some penalty mechanism to be applied to the misbehaving parties [19]. However, it is important to note that from a practical perspective, this advantage could be small enough in order to be tolerated by both parties.

In [11] Bitcoin is used for the payment in an optimistic fair exchange (with a TTP) with anonymity. Most notably, Bitcoin has been proposed for fair exchange as a mean of implementing a penalty mechanism [5]. The idea is that if a party leaves the protocol with more knowledge than the rest, those honest parties are compensated. The same idea is applied to multiparty computation in [1].

In our proposal Bitcoin is used as the main mechanism to implement and enforce the fair exchange. We do not rely in the use of a TTP and instead of fairly sign a contract for the exchange of data, the contract is explicitly executed as a Bitcoin smart contract. This has the advantage that the exchange is produced, that is the buyer gets the data and the seller the money, completely or not. In some sense we can say that the exchange is atomic. Although our approach does not achieve strong fairness, as we will show, the advantage that a party (in our case the buyer) takes by leaving the protocol can be bound by the other party.

Similarly to our proposal, [6, 12] uses Bitcoin with zero knowledge proofs to allow payments in Bitcoin subject to the disclosure of a given secret. In this case the secret is a symmetric key used to encrypt some given data. A zero knowledge proof is used (externally to Bitcoin) to prove the validity of the encrypted data and the secret key, thus providing some sort of strong fair exchange. This is only feasible if a zero knowledge proof exists and is feasible for the specific case (data). A more generic solution is outlined in [18, 17] where a symmetric key is used to encrypt chunks of data such that a subset can be revealed as a proof. As different keys are used for each chunk, revealing a subset does not ensure that the key of the other chunks is correct. In our proposal, since the key revealed is a private key, it is easy to verify its correctness with the corresponding public key. This ensures that the key is valid for all data chunks by just verifying one of them.

3 Bitcoin transactions background

Bitcoin transactions are usually seen as a transfer of bitcoins from a source to a destination address, in which the former can prove the ownership of such bitcoins, and thus spend them, by providing a digital signature. However, Bitcoin transactions are far more complex, and allow the creation of richer conditions that have to be met to redeem the funds [14]. Transactions may be seen, in a more general way, as a collection of inputs, containing references to previous transaction outputs, and a collection of outputs with their corresponding conditions under which they will be able to be redeemed. Each input of a transaction refers to an output of a previous one, where unlocking conditions have been established. Hence, each input has to contain the proof of fulfillment of the established conditions of the output he tries to redeem from.

Both unlocking conditions and proofs are coded in transactions using *Script*, a stack-based, not Turing-complete scripting language with no loops. In order to check the correctness of a transaction, the full script is executed by concatenating both locking and unlocking scripts, leading to a final `True` popped up into the top of the stack if and only if the proof satisfies the unlocking conditions. This kind of transactions including complex unlocking scripts are also known as smart contracts. However, not every single condition can be coded nor checked using `Script`. A limited number of operations (opcodes) are defined in Bitcoin, bounding the variety of scripts that can be encoded using the language. Furthermore, its use is even more restricted, since not all the defined operations can actually be used. For instance, the most common script transaction within Bitcoin, the standard Pay-To-Public-Key-Hash where a digital signature is needed to redeem a transaction, is next provided:

```
ScriptPubKey:  OP_DUP OP_HASH160 <pubKeyHash> OP_EQUALVERIFY
                 OP_CHECKSIG
ScriptSig:     <sig> <pubKey>
```

Based on the Bitcoin scripting language, multiple special-purpose transactions can be defined. In the following subsections some interesting types of transactions, that are building blocks of the proposed data trading protocol, are described.

3.1 Time locked transactions

Time locked transaction outputs are outputs that require a certain time in the future to be reached in order to be redeemed. Depending on whether the future time is absolute to Bitcoin, or relative to the transaction publishing time, two types of time-locks can be found. On the one hand, absolute time-locks, those based on the `CheckLockTimeVerify` opcode, establish a fixed date in the future from when the transaction can be redeemed. Down bellow an example of such time-lock (locked until 2022/12/13), along with a standard signature, is provided:

```
ScriptPubKey:  <2022/12/13> OP_CHECKLOCKTIMEVERIFY OP_DROP
                 <pubKey> OP_CHECKSIG
ScriptSig:     <sig>
```

On the other hand, relative time-locks, those based on the `CheckSequenceVerify` opcode, establish an amount of time, starting from the transaction publishing time, that has to be spent to unlock the output. An example of such time-lock (locked for 25 days), together with a traditional signature, can be found as follows:

```
ScriptPubKey:  <25d> OP_CHECKSEQUENCEVERIFY OP_DROP
                 <pubKey> OP_CHECKSIG
ScriptSig:     <sig>
```

Notice that in both examples the `ScriptSig`, included in the transaction that will spent the output, does not contain any time reference, since the transaction creation time is used to check the time-locks. Moreover, both examples include a traditional signature lock. The reason behind this second lock is to restrict the redeemer to a single person, otherwise anyone will be able to spend the output once the requested time has been reached. Figure 1 depicts a general time locked transaction, and can be seen as a representation of any of the two introduced types.

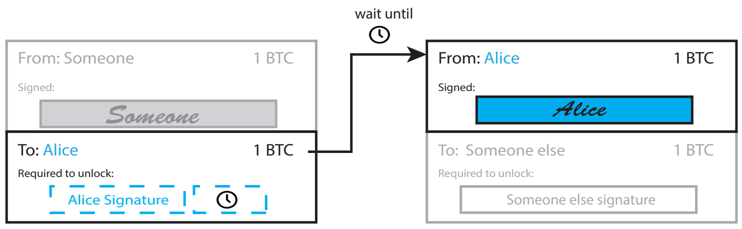

Figure 1: Time locked transaction.

3.2 Private key locked transactions

Private key locked transactions [8] are another special case of Bitcoin transactions in which the transaction output can be redeemed by anyone who provides a private key corresponding to a public predefined key.

Two different approaches can be used to implement private key locked transactions, via the definition of new Bitcoin opcode or by taking advantage of a well-known vulnerability of ECDSA algorithm.

3.2.1 New Bitcoin opcode

One possibility to implement private key locked transactions is through the implementation of a new crypto opcode that performs precisely a matching validation between the public key and the corresponding private key: `OP_CHECKKEYPAIRVERIFY`. The `OP_CHECKKEYPAIRVERIFY` opcode would check whether the top two items of the stack, `pubKey` and `privKey` (corresponding, respectively, to a public key and private key), match.

With the usage of this new opcode, a transaction output could be constructed such that, in order to be redeemed, the private key matching the specified public key has to be revealed. An example[1] of the `scriptPubKey` of such an output together with the `scriptSig` needed to spend it would be:

```
ScriptPubKey:  <pubKey> OP_CHECKKEYPAIRVERIFY OP_NIP OP_CHECKSIG
ScriptSig:     <sig> <privKey>
```

The script will first check that the public and private keys belong to the same key pair. Note that, if the validation is successful, the stack values will remain untouched. Therefore, before checking the validity of the signature with `OP_CHECKSIG`, the `privKey` value has to be removed from the stack (since it is not needed for signature validation). The execution of `OP_NIP` removes `privKey` from the stack. Finally, `OP_CHECKSIG` validates the signature with the public key. If the signature is correct, the script terminates successfully.

Note that the execution of `OP_CHECKKEYPAIRVERIFY` would fail if the validation is unsuccessful and would leave the stack as it was before if the validation is successful. This ensures that the new opcode can be implemented as a soft fork modification of the Bitcoin core protocol by reusing one of the currently unused `OP_NOPx` opcodes, in a similar way that it has been done in the past with `OP_CHECKLOCKTIMEVERIFY` (`OP_NOP2`) and `OP_CHECKSEQUENCEVERIFY` (`OP_NOP3`).

3.2.2 ECDSA vulnerability

Since the `OP_CHECKKEYPAIRVERIFY` opcode described in the previous section is not still available, another approach to build transaction outputs that require

[1]The provided script includes a digital signature condition, following the structure of the ones previously introduced in Section 3.1.

the disclosure of a specific private key to be redeemed can be taken, using a vulnerability in the ECDSA signature scheme.

ECDSA (Elliptic Curve Digital Signature Algorithm) is the cryptographic algorithm used by Bitcoin to create and validate digital signatures. The ECDSA signature scheme is probabilistic in the sense that there exist many different valid signatures made with the same private key for the same message. Such feature is based on the selection of a specific random value k during the signature process.

There exists a well known ECDSA signature vulnerability[2] by which an attacker that observes two signatures of different messages made with the same private key is able to extract the private key if the signer reuses the same k during the signature process. Therefore, the selection of k is critical to the security of the system.

To implement private key lock transactions, we can make use of the aforementioned ECDSA vulnerability to perform targeted private key disclosure within Bitcoin. The Private key disclosure mechanism is performed by constructing transaction outputs that need to reveal a private key in order to be redeemed, in such a way that we ensure the revealed private key is the counterpart of a certain public key.

Let $\{PK, SK\}$ be an ECDSA key pair belonging to Bob (with $Addr(PK)$ the Bitcoin address associated to it) and sig_{prev} an existing signature made with SK. Alice (that is interested in acquiring Bob's private key) needs to know the value of the previous signature sig_{prev}, in order to be able to request, afterwards, a second signature made with the same k. In contrast with the approach followed in [8], where the previous signature appears in the blockchain as the input script of an existing transaction, in this paper our approach is that the existing signature sig_{prev} does not appear in the blockchain but it is sent to Alice by an off-chain exchange of values. In this case, the previous signature sig_{prev} may be transmitted confidentially (and thus only Alice and Bob know its value). Following this approach, the signed message m does not need to correspond to a Bitcoin transaction hash.

Once an existing previous signature sig_{prev} is known by Alice, she creates a transaction with an output that requires a second signature sig to be spent. However, instead of using the classical pay-to-pubkey-hash script, she uses a special script that forces Bob (the redeemer) not only to prove he has the private key SK associated to the given address $Addr(PK)$ by creating a valid signature, but also to deliver a signature that has exactly the same k value that was used when creating sig_{prev}.

Doing so accomplishes two purposes: on the one hand, Bob proves he knows the private key associated to the public key by generating a signature that correctly validates with that public key; on the other hand, Bob is implicitly revealing the private key associated to the same public key. Note that Bob does not directly provide the private key, but provides information from which the private key can be derived.

[2]The vulnerability is also present in the non-elliptic curve signature scheme of ElGamal (and its popular variant, DSA) and is described in any fundamental cryptography text book [13, 15].

Figure 2 shows an scheme of the Bitcoin transactions involved in the construction of a a private key locked output. Once Alice knows the previous signature, she can construct the transaction Tx_2, that transfers some bitcoins of her property to Bob, only if Bob provides a valid signature that has the same k as the previous signature sig_{prev} that Bob previously sent to Alice. Moreover, the output has an additional condition with a time lock allowing Alice to get a refund of her bitcoins if Bob decides not to collaborate and does not redeem Tx_2's output.

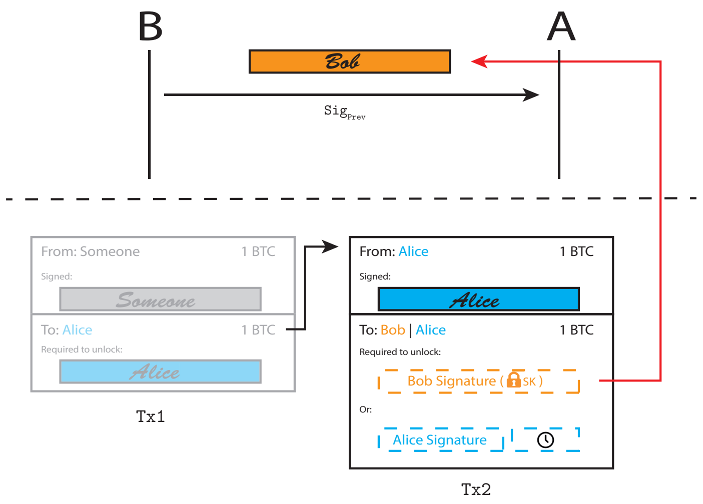

Figure 2: Transactions involved in the scheme.

The `ScriptPubKey` of the output (and its corresponding `ScriptSig`) that implement the mechanism described above are the following:

```
ScriptPubKey:  OP_DUP <pubKey> OP_CHECKSIGVERIFY
                 OP_SIZE <0x47> OP_EQUALVERIFY
                 <sigmask> OP_AND <kprev> OP_EQUAL
ScriptSig:     <sig>
```

First, the script validates the signature against the specified public key. Then, the length of the signature is checked. Finally, a bitwise AND between the new signature and sig_{mask}[3] is computed, and the result is compared with the k value of the previous signature. If both values are equal, the script terminates successfully; otherwise, the script terminates with a False value on the stack, making it fail.

Note that the only way to ensure that the script succeeds is by providing a

[3] sig_{mask}: a byte array that has 1s on selected positions and 0s in the rest of positions in order to be able to extract information from the k value of the signature (see [8] for more details).

valid signature that has exactly the same k as the previous signature. Therefore, although the redeem `ScriptSig` that spends the output does not include the value of the private key directly, it is implicitly leaking its value by the ECDSA vulnerabilty.

Also note that the `ScriptSig` needed to spend the output only requires one value: the new signature.

4 The data trading protocol

As we have already stated, the main property of our data trading protocol is its atomicity which provides its fairness for the participants of the protocol. The proposed protocol is run by two parties, the buyer, B, and the seller, S, and no additional party, like a TTP is needed. Notice that, contrary to other fair exchange protocols [2, 21, 3], our proposal does not define a dispute mechanism, in which some proposals also need a TTP, thanks to the atomicity of the protocol. Furthermore, since our protocol is based on bitcoins, both parties need to be connected with the Bitcoin network to send/receive transactions from the blockchain.

In our scenario, the buyer, B, wants to buy some data D to the seller, S, and he is willing to pay x bitcoins for such data. In order to minimize any possible advantage of one of the parties, we consider that the data being sold can be divided in n different parts and each of those parts may have a meaning by itself. Notice that this scenario is not as restrictive as it would appear since multiple data falls into this category. For instance, movies or songs can be sliced and each slice may be recognized as part of the whole performance. On the other hand, when dealing with sensor data, some sensing values may provide evidence that the sensing is correct but the whole sensing data could be needed for specific purposes.

4.1 Protocol description

The full protocol, depicted in Figure 3, can be divided in three main parts: the *Data correctness proof*, in which a cut & choose protocol between B and S is performed in order to convince B that the acquired data is correct. The *Signature commitment*, used for B to obtain a previous signature performed by S with the private key used to encrypt the data. And the *Private Key Exchange*, used to exchange, atomically, the private key that allows to decrypt the sold information for the agreed amount of bitcoins.

In the following paragraphs, we describe in detail each subprotocol. We denote by $\{PK, SK\}$ a public key pair and $E_{PK}(\cdot)$ the encryption function using the public key.

In the **Data correctness proof** subprotocol, the buyer B starts the protocol by requesting data to the seller S. In such first step, B will indicate to S the data he is wiling to buy. Such request will include the conditions, *cond*, that the data being sold has to hold. Such conditions need to be hold not only for the

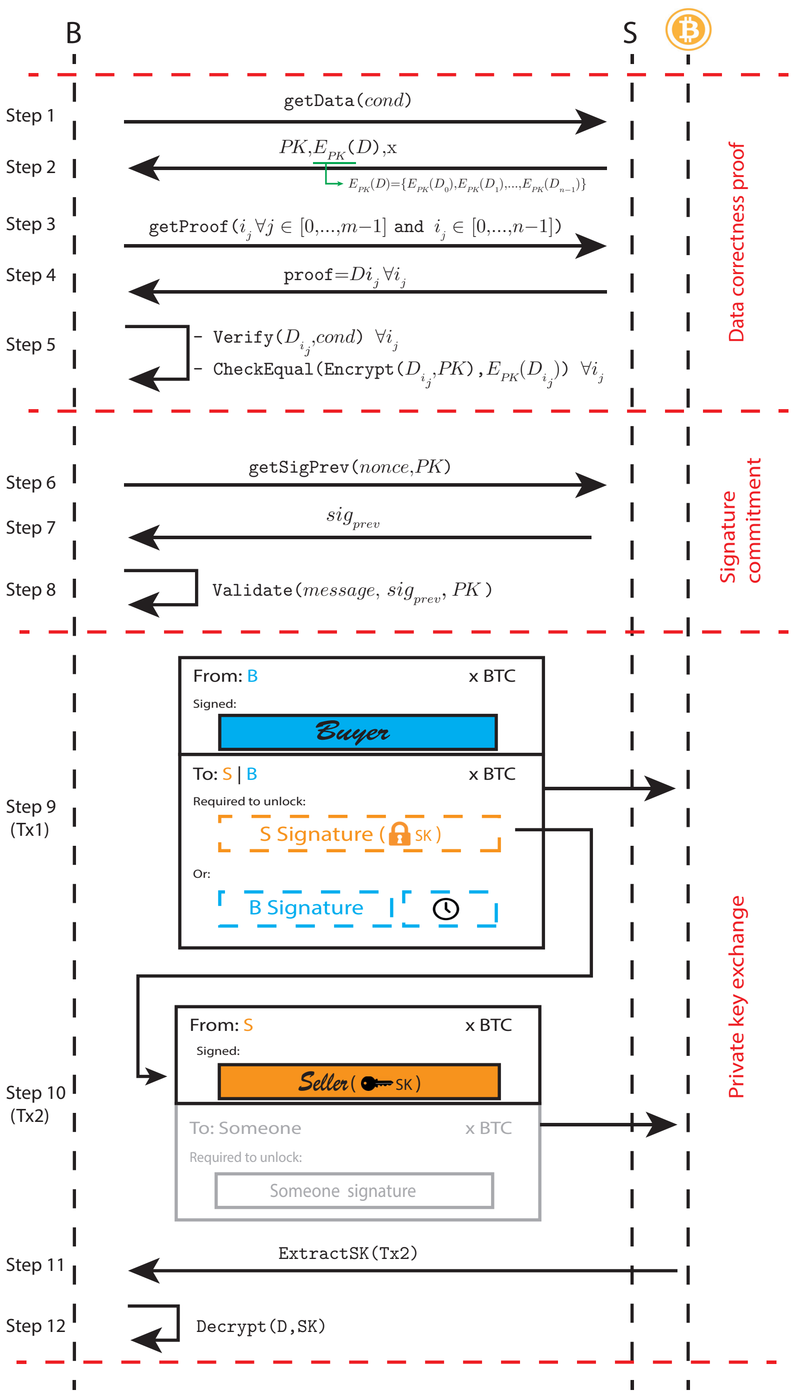

Figure 3: Fair data trading protocol.

complete data being sold but also for each of the sliced part of the data[4]. Upon reception, S generates a new key pair $\{PK, SK\}$ and sends B the following information (see Step 2 in Figure 3): the public key PK, the requested data D encrypted using PK, and the data price x. In order to allow B to prove the data correctness, S does not send the D as a whole bunch of encrypted data, but split in n chunks which are encrypted individually (as shown in Figure 4), that is: $E_{PK}(D) = \{E_{PK}(D_0), E_{PK}(D_1), ..., E_{PK}(D_{n-1})\}$.

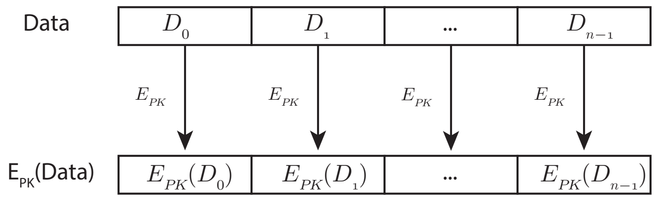

Figure 4: Split and encrypt procedure.

When B receives all the encrypted data, he requests a correctness proof to S consisting in a random subset of non-encrypted data from D. To that end, B selects the subset by randomly choosing a set of m pieces from the encrypted dataset, that is $i_j \;\; \forall j \in [0, m-1], \;\; i_j \in [0, n-1]$. B sends this information and S can build the correctness proof by choosing the unencrypted pieces of data that matches the received indexes, that is, $proof = \{D_{i_j} \;\;\; \forall j \in [0, m-1], \;\; i_j \in [0, n-1]\}$. S sends such correctness proof to B.

Once B has received the $proof$ he verifies the correctness of D by checking that the proof satisfies the conditions. Furthermore, B validates that the received data also matches with the subset of received encrypted data, by recreating the data encryption using PK. Therefore, since the subset has been randomly chosen by B, the correctness of the full dataset can be proved with a given probability. Section 4.3 analyzes in depth the impact of the parameters of the scheme on such probability.

Once the data correctness has been proven, the **Signature commitment** subprotocol is performed. B requests a signature Sig_{prev} over a nonce message performed with the private key SK generated by S. S sends Sig_{prev} and B validates that the signature is correct, using the public key PK that has received in Step 2 of the Data Correctness Proof subprotocol.

Finally, the **Private Key Exchange** subprotocol is performed. In such subprotocol, B builds a private key locked transaction, Tx_1, to perform the atomic exchange between the private key, SK, and the bitcoin price x. Such private key locked transaction is built using the technique described in Section 3.2 and also adding another time constrain condition following the details of Section 3.1. Such time constrain is used for B to recover the amount of x bitcoins in case S decides not to reveal the private key by not spending the received transaction.

[4]Notice that such conditions will be verified by a validation mechanism. Whether such mechanism is performed automatically or the validation needs a supervised environment is out of the scope of our protocol.

B broadcasts the transaction Tx_1 to the Bitcoin P2P network. Once Tx_1 is included in a block, S can spend the output of such transaction with an input of a new transaction Tx_2 in which S will provide the second signature with the same k of sig_{prev}. Once Tx_2 appears on the blockchain, B will be able to recover the private key SK (following the details of Section 3.2) and decrypt the data $E_{PK}(D)$ he received in Step 2 to retrieve the purchased data.

4.2 Implementation details

In the protocol description provided in the previous section some implementation details have deliberately omitted to allow a better understanding of the general protocol. In this section, we provide some comments regarding such specific details.

4.2.1 Privacy protection

First of all, sensitive information exchanged in the protocol should be protected from third parties. For instance, if an attacker could retrieve the information transmitted in Step 2 and in Step 7, later on, with the knowledge of Tx_2, which is publicly available in the blockchain, he could decrypt the information and retrieve the original data D. To avoid such situation, information transmitted on steps 2 and 7 could be encrypted using the public key of B, that could be sent to S in Step 1. Furthermore, in Step 4, some part of the data is transmitted in clear for validation purposes. In this case, an external attacker could also obtain such information. Again, such situation can be avoided by encrypting the information of Step 4 in the same way we just described for Steps 2 and 7.

4.2.2 Data encryption mechanism

As it is well known, public key cryptography is not suitable for encrypting large files due to its poor performance. Then, since the size of the data chunks that are encrypted and transmitted in Step 2 can vary depending of the traded data, we suggest to use digital envelopes to encrypt D. Digital envelopes [13] protect the message by using a two layer encryption in which the data itself is encrypted using symmetric encryption, and then the symmetric key is encrypted using public-key cryptography. Following such an approach, for each chunk i of data created from D, D_i, a symmetric key k_i is also generated. D_i will be encrypted using k_i, that is $C_i = E_{k_i}(D_i)$ and k_i will then be encrypted using PK, that is, $c_i = E_{PK}(k_i)$. Thus, each encrypted chunk of data D_i sent by S to B during Step 2 should be replaced by $\{C_i,\ c_i\}$, that is, $E_{PK}(D_i) \rightarrow \{C_i,\ c_i\}$. Furthermore, when sending the correctness proof, S will include the corresponding symmetric encryption keys $k_i\ \forall i \in 0, ..., m-1$. Finally, B will need to undo the digital envelope process in Step 5 in order to perform all the required verifications, and also in Step 12, when finally decrypts D.

4.2.3 Script building

The private key locked transaction used in the secret key exchange subprotocol, Tx_1 also includes a time lock condition to allow B to refund his x bitcoins in case S decides not correctly follow the last step of the protocol. The details on how the `ScriptPubKey` of such transaction can be build are next provided [5] :

```
ScriptPubKey: IF
                  OP_DUP <B pubKey> OP_CHECKSIGVERIFY
                  OP_SIZE <0x47> OP_EQUALVERIFY
                  <sigmask> OP_AND <rprev> OP_EQUAL
              ELSE
                  <expiring time> OP_CHECKLOCKTIMEVERIFY
                  OP_DROP <S pubKey> OP_CHECKSIG
              ENDIF
```

4.3 Protocol fairness discussion

The main objective of the proposed protocol is to achieve fairness in the sense that neither B nor S would have any advantage in the protocol. By advantage we mean that B cannot obtain the data without paying x bitcoins and S cannot obtain the bitcoins without revealing the data. The Data Correctness Proof subprotocol ensures that S cannot sell fake data. Without B verifying parts of the encrypted data, S could encrypt fake data and when S obtains the bitcoins in Tx_2 and B the decryption key, B will learn that he was cheated but it will be too late since S already has the bitcoins.

In the following paragraphs, we will show how the buyer B is probabilistically protected against deception by using a cut-and-choose mechanism. Furthermore, the level of protection may be adjusted by fixing the ratio of chunks revealed on Step 2.

In the main steps of the Data Correctness Proof subprotocol:

1. S encrypts each of the n chunks of data with a public key PK and commits to the ciphered chunks by sending them to B.

2. B chooses a subset of m chunks and asks S to reveal the original data corresponding to those chunks.

3. B validates the received chunks by checking both that the original data meets the specified conditions and that the encryption of the original data is equal to the committed values.

We say that a seller S successfully deceives a buyer B if the seller is able to include b corrupted chunks of data within the n traded chunks without the buyer

[5] An example of such a transaction can be found in
`http://tbtc.blockr.io/api/v1/tx/info/c1ecc2d4bad00e65aab425071ced1eb7dbdeeb1b3712a3df2d4dccd49c907f09`

noticing it after having validated the m revealed chunks (that is, after finishing the Data Correctness Proof subprotocol). The probability Ω of S successfully deceiving B is given by the following equation:

$$\Omega(m,n,b) = 1 - \sum_{i=1}^{\min\{b,m\}} \frac{\binom{b}{i}\binom{n-b}{m-i}}{\binom{n}{m}}$$

Indeed, $\binom{n}{m}$ counts the number of ways of choosing m elements from a set of n elements. We are interested in knowing how many of those ways include at least one corrupted chunk. We compute this value by counting the number of ways of selecting m elements with exactly i of them being corrupt and summing them up for all possible i values. $\binom{b}{i}\binom{n-b}{m-i}$ computes the number of ways of selecting exactly i corrupted chunks, that is, the number of ways of selecting i bad chunks from the set of b corrupted chunks, $\binom{b}{i}$, multiplied by the number of ways of selecting the rest $m-i$ elements from the non corrupted set $n-b$, $\binom{n-b}{m-i}$. The summation gives the probability of selecting at least one corrupted chunk within the m reveleaded, that is, the probability of detecting a fraud. Therefore, the probability of deception is the complement.

Figure 5 shows the probability of deception Ω for different ratios of chunks revealed, $\frac{m}{n}$, and different number of corrupted chunks included by the seller, b, for $n = 1\,000$. Note that, even when the checked chunks ratio is low, the probability of successfully deceiving a buyer is low whenever b is over a certain threshold. For instance, when 20% of the chunks are checked, the probability of deception is 0.8 if the seller includes just 0.1% of corrupted chunks ($b = 1$, red dot on Fig. 5). However, if the seller includes 1% of corrupted chunks ($b = 10$), the probability of successfully deceiving the buyer decreases to 0.106 (green dot on Fig. 5).

When using the proposed data selling protocol, the two parties (buyer and seller) agree on the value of the parameter m. Therefore, the buyer can decide beforehand whether or not to buy a given dataset depending on the deception risk he is willing to assume. Buyers will be interested in using high m values, since these offer higher levels of protection. Of course, even honest sellers will prefer low m values, since if the client does not finally buy the data, m data chunks end up being revealed for free.

As an example, let us consider a certain seller S that sells a dataset divided in $n = 1\,000$ chunks. The buyer may allow to accept, at most, 5% of corrupted chunks in his purchased data, and the seller does not want neither to reveal in the Step 2 more than the 5% of the chunks. With this configuration, if we analyze the probability Ω for $b = 50$ (the 5% of the $1\,000$ chunks),

$$\Omega(50, 1\,000, 50) \approx 0.072$$

the buyer can be sure that the seller cannot cheat with a probability greater $1 - 0.072 = 0.928$. Of course, if the buyer does not have any trust in the seller, he could force the seller to reveal 10% of chunks instead of only 5% in Step

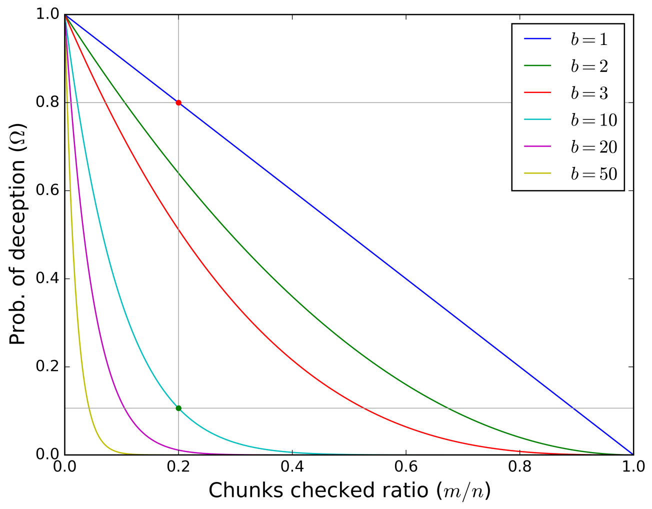

Figure 5: Probability of deception Ω. Grey lines highlight the values mentioned as numerical examples.

2. With this settings, the buyer knows that the probability the seller cheats is almost negligible, less than a 0.004.

5 Conclusion

In this paper we have introduced a fair data trading protocol based on Bitcoin transactions. The protocol uses a new type of transactions, the private key locked transaction, that provide an atomic way of exchanging a private key for Bitcoins. Such key is used to encrypt all the traded data, and will be traded, as a part of a Bitcoin smart contract, only when the two parties agree. The correctness of the data sold using the protocol is verifiable by the buyer before performing the transaction by checking a small random subset of data. By using such a cut-and-choose technique, deception is avoided with a high probability while only a small part of the information is learned by the buyer.

The protocol can be implemented by using the recently proposed private key locked transaction and exchanging a few messages between the parties involved in the process, making it easy to deploy. Moreover, it lays on the security measures Bitcoin provides, without introducing more complexity, and it is bound to the computational capabilities of the Bitcoin Scripting language.

We believe that, since there is no need of any other entity, like a TTP, to implement the protocol, it could be easily deployed to provide an additional security layer in the process of data trading using Bitcoins, reducing the trust the involved parties have to share among them, and promoting the use of such a currency for trading with non-physical goods. The application of such a protocol covers a wide range of topics, from general interest data, such as songs, pictures or movies, to even specific purpose data, such as data sensing readings. The integration of such a data trading in data sensing scenarios could provide a secure way of data correctness verification, reducing users misbehaving ratio and optimizing the rewarding system.

Acknowledgements

This work is partially supported by the Spanish ministry under grant number 785 TIN2014-55243-P and the Catalan AGAUR grant 2014SGR-691. Authors would like to thanks Aaron van Wirdum from the Bitcoin Magazine for his illustrations of Bitcoin transactions, in which we have based to create the ones provided along the paper. Also, authors would like to thank our colleague Roger Ten-Valls for his help on the discussion during the calculation of the deception probability equation involved in the protocol.

References

[1] Marcin Andrychowicz, Stefan Dziembowski, Daniel Malinowski, and Lukasz Mazurek. Secure multiparty computations on bitcoin. *Commun. ACM*, 59(4):76–84, March 2016.

[2] N. Asokan, V. Shoup, and M. Waidner. Optimistic fair exchange of digital signatures. *IEEE Journal on Selected Areas in Communications*, 18(4):593–610, April 2000.

[3] Feng Bao, R. H. Deng, and Wenbo Mao. Efficient and practical fair exchange protocols with off-line ttp. In *Proceedings. 1998 IEEE Symposium on Security and Privacy (Cat. No.98CB36186)*, pages 77–85, 1998.

[4] M. Ben-Or, O. Goldreich, S. Micali, and R. L. Rivest. A fair protocol for signing contracts. *IEEE Transactions on Information Theory*, 36(1):40–46, January 1990.

[5] Iddo Bentov and Ranjit Kumaresan. How to use bitcoin to design fair protocols. In Juan A. Garay and Rosario Gennaro, editors, *Advances in Cryptology – CRYPTO 2014: 34th Annual Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2014, Proceedings, Part II*, pages 421–439. Springer Berlin Heidelberg, 2014.

[6] Bitcoin Wiki. Zero knowledge contingent payment. `https://en.bitcoin.it/wiki/Zero_Knowledge_Contingent_Payment`, February 2016.

[7] M. Blum. How to exchange (secret) keys. *ACM Trans. Comput. Syst.*, 1(2):175–193, May 1983.

[8] Sergi Delgado-Segura, Cristina Pérez-Solà, Jordi Herrera-Joancomartí, and Guillermo Navarro-Arribas. Bitcoin private key locked transactions (short paper). Cryptology ePrint Archive, Report 2016/1184, 2016. `http://eprint.iacr.org/2016/1184`.

[9] Shimon Even, Oded Goldreich, and Abraham Lempel. A randomized protocol for signing contracts. *Commun. ACM*, 28(6):637–647, June 1985.

[10] Matthew K. Franklin and Michael K. Reiter. Fair exchange with a semi-trusted third party (extended abstract). In *Proceedings of the 4th ACM Conference on Computer and Communications Security*, CCS '97, pages 1–5, 1997.

[11] D. Jayasinghe, K. Markantonakis, and K. Mayes. Optimistic fair-exchange with anonymity for bitcoin users. In *2014 IEEE 11th International Conference on e-Business Engineering*, pages 44–51, 2014.

[12] Gregory Maxwell. The first successful zero-knowledge contingent payment. Bitcoin Core Blog, `https://bitcoincore.org/en/2016/02/26/zero-knowledge-contingent-payments-announcement/`, February 2016.

[13] Alfred J Menezes, Paul C Van Oorschot, and Scott A Vanstone. *Handbook of applied cryptography.* CRC press, 1996.

[14] Arvind Narayanan, Joseph Bonneau, Edward Felten, Andrew Miller, and Steven Goldfeder. *Bitcoin and cryptocurrency technologies.* Princeton University Pres, 2016.

[15] Christof Paar and Jan Pelzl. *Understanding cryptography: a textbook for students and practitioners.* Springer Science & Business Media, 2009.

[16] Indrajit Ray and Indrakshi Ray. Fair Exchange in E-commerce. *SIGecom Exch.*, 3(2):9–17, March 2002.

[17] Peter Todd. OP_CHECKLOCKTIMEVERIFY. BIP 65, `CHECKLOCKTIMEVERIFY`, 2014.

[18] Peter Todd and Amir Taaki. Paypub: Trustless payments for information publishing on bitcoin. Github Project, `https://github.com/unsystem/paypub`, October 2004.

[19] T.W. Sandholm and V.R. Lesser. Advantages of a leveled commitment contracting. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence and Eighth Innovative Applications of Artificial Intelligence Conference, AAAI 96, IAAI 96,*, volume 1, pages 126–133, 1996.

[20] Jianying Zhou and D. Gollman. A fair non-repudiation protocol. In *Proceedings 1996 IEEE Symposium on Security and Privacy*, pages 55–61, May 1996.

[21] Jianying Zhou and D. Gollmann. An efficient non-repudiation protocol. In *Proceedings 10th Computer Security Foundations Workshop*, pages 126–132, 1997.