Fair exchange in the blockchain

Draft

A W Roscoe, Bangdao Chen and Wang Lei*
Chieftin Lab, Shenzhen

July 24, 2018

Abstract

Fair exchange is an important primitive in transactions where the parties do not completely trust one another. It is well known to be impossible to implement this with 100% assurance without a trusted third party. The blockchain is a sort of trusted third party built out of many untrusted parties, and a very popular target for systems involving peer-to-peer (i.e. without passing via central authorities) transactions. In this paper we show a number different ways in which fair exchange can be implemented.

1 Introduction

Suppose a group of parties have agreed a transaction in principle that will involve them passing valuable things to one another in some pattern. We assume that all these things, whether goods or money, can be passed from one party to another via a single electronic message passing a digital token. Even if they agree a complete schedule of how the value will be passed, there is nothing to compel any of the group to send a particular message. The danger is that after Alice has sent Bob X, there is nothing to compel Bob to send Alice Y in return. Of course there may be mechanisms for putting Bob in disgrace, but there is no direct way of preventing him from running away with both X and Y. We will concentrate on binary exchange, namely between two parties.

There is an extensive literature on fair exchange, including the demonstration that implementing it with 100% guarantees is impossible. Two

*Also University College Blockchain Research Centre, Oxford

approximations to this are *optimistic* fair exchange, in which the parties go ahead without a TTP, but can bring him in to complete the deal if all does not happen properly, and *stochastic* fair exchange, in which the exchange happens at a point and order unknown to either party. If one of them cheats then he does not know whether he is benefiting himself or the other party, and we can make the likelihood of these as nearly equal as we please.

The blockchain is widely proposed as a platform in which parties can hold assets and trade them. The standard unit of commerce in the blockchain is a transaction that passes value from one party to another: value is an unspent transaction to your identity, as represented by a public key, and you pass value on to another party by signing a copy of the transaction, details of the identity receiving it, and the amount being transferred. Thus the problem of fairness remains: Bob can still receive a transaction from Alice and omit to send one back.

In this paper we make the blockchain's participants co-operate to generate essentially the same sort of behaviour that we would expect a traditional TTP to have: there is no expectation that both transactions will happen in the same blockchain or any.

2 Background

It was proved in [] that binary fair exchange is impossible without a trusted third party. The blockchain architecture is designed for distributed applications in which there is no universally trusted party. At first sight it therefore seems as though fair exchange ought to be impossible in such a scenario. The blockchain is, however, a structure designed to create a reliable distributed database in the context where some majority of parties in it (typically just a simple majority, perhaps with some weighting), are trustworthy.

A blockchain is simply a chain of blocks of data, linked together by each holding the cryptographic hash of its predecessor. It is coupled with a protocol for adding new blocks. The parties who create new blocks are generally called *miners*. In a public blockchain anyone is allowed to propose a new block, whereas in a private or "coalition" one only an authorised party may do so. In the latter the assumption is usually that the collective trustworthiness of these miners is understood and acknowledged by all, whereas in public models there are costs and incentives build in to the mining process which are designed to incentivise honest and accurate mining.

The objective is that once a block is in the chain it will, sooner or later, become fixed, or *immutable*: it will never change. The immutable

part of a blockchain is an attractive data abstraction because it provides a deterministic communication mechanism between the parties and an agreed truth. Blockchains are write-only databases: data is only added to. The index of a block that some item is provides a reliable time-stamp, and this time-stamping mechanism can be refined by adding time-stamps to items within blocks in a necessarily (because of block agreement) agreed way.

Blockchains are well-known for their ability to support transaction mechanisms such as digital currencies, but have many other proposed uses.

In any blockchain we really need to be able to identify the time at which any block becomes immutable: will remain on the chain for ever together with all its predecessors. Such a time might be explicit and in many private blockchains will equate to the point at which a block is created and approved. In others there might be a mechanism for defining this point, for example by the open publication (e.g. in a news paper) of the hash of this or a successor block. In further ones it may simply be because the cost of revising the block and necessarily its successors has passed some limit of practicality.

Any form of transaction which is either between a blockchain and the rest of the world, or in which a blockchain mediates a transaction involving value that lies entirely outside itself, needs to bear this question of immutability in mind. For it does not make sense to swap something real and tangible for a transaction which might subsequently get voted out of history. Thus we are free to assume that fair exchange wholly within a blockchain can live within the standard understanding of blocks there, but where we are using one to determine external transactions, we will assume that data must be immutable before it is acted on.

The methods described in this paper are best suited to private and mainstream (hybrid) models of blockchain where there are parties who are motivated to behave in a trustworthy manner for reasons other than that there is no profit to be made from not doing so.

3 Fair exchange via distributed escrow

In a traditional setting, after Alice and Bob both agree a transaction: Alice sends X for Bob in return for Y, they sign a certificate about what they intend to do and send it to a TTP. They then send X and Y to the TTP, who when it has the certificate, X and Y sends X to B and Y to A. If the TTP does not get these things it sends whichever of X and Y it holds back to sender. We can also give the TTP the duty of checking that X and Y have whatever properties are claimed of them in the contract.

In the blockchain no single party is trustworthy, so we need to find a way of making the nodes as a whole behave like the TTP above. We certainly cannot give any single party control over X or Y. In essence we want the nodes of the blockchain to hold X and/or Y in something like an escrow form and release them to both the recipients or none at the point where it has them both. We will refer to the nodes that perform the transfer as the *workers*. In some contexts it would make sense to identify these with the miners of the blockchain.

We will assume that X and Y take the form of bit-strings that pass value: something like an as yet unregistered blockchain transaction, where Bob and Alice placing them on the blockchain will transfer value to them. We will always assume that A and B have signed a certificate of intent to exchange X and Y, and this is visible to all the workers. We observe that it is not often that simple knowledge of a string of bits conveys value. One example of this would be the number of a bank account from which anyone knowing the number could obtain money.

More usually the string is something that identifies Bob (or at least an alias) and transfers the asset to him: perhaps a cheque, stock certificate in his name that he has to register, or a property transfer document. In the blockchain a transaction would identify the agent that the asset is being transferred to.

The difference between these two cases is that in the first we cannot allow a possibly untrustworthy third party to know the bit-string, whereas in the second we can (unless secrecy of the transaction or some detail of it is itself required.) For the time being we will assume that the intermediaries may not know X and Y, and later explore another option when they can.

Following on from [**?**] (which shows how to model delay encryption in a blockchain-like environment where trust is again not placed in any one party) a good first move seems to be for Alice and Bob to apply threshold cryptographic construction to X and Y so that holding less than k shares conveys no information about them, and holding k or more conveys the complete value. We will discuss later what N and k might be, but certainly N is less than or equal to the number of participants in the blockchain available to be workers. If A and B are themselves in the pool of potential workers, it is probably a good idea to prevent them from being chosen.

We will assume that each participant has its own public and secret keys that can be used for sending it confidential information and allowing it to sign things.

As a first attempt assume that each participant behaves, with threshold shares, like the single TTP described above. It waits to get shares of X and

Y and, when it has one of each, forwards these to Bob and Alice respectively. This does not work, for if Alice sends just k shares and Bob sends all his, but at least one of the parties receiving one of Alice's shares is corrupt and under Alice's control, then Alice can receive k shares of Y (and thus Y itself) but the corrupt one will not send its X share to Bob, meaning that Bob does not receive Y.

We can, however, solve this problem using the properties of the blockchain as a data structure that gives a consistent view to everyone. Modify the above protocol so that as soon as any worker has a share of X and one of Y it posts a signed message to this effect (but not the shares) on the blockchain: call these the intermediate certificates. Note that if $2k-1$ nodes have posted such a message then at least k of them must be telling the truth, because there are less than or equal to $k-1$ corrupt ones. They therefore take the existence of $2k-1$ such signed messages on the blockchain as a trigger to release their shares to Bob and Alice. These parties must now receive at least k valid shares of X and Y.

We can ensure that they do not accept any incorrect shares because Alice and Bob will be told to place a hash of each share on the blockchain at the start of this protocol. Thus if Bob or Alice (or indeed anyone else) receives something purporting to be a share which is not, this can be checked. (Alternatively Alice and Bob can simply sign all the shares.) We will later discuss how that X and Y that are being traded can be checked rather than simply that Alice and Bob are sending known shares.

It follows that the existence of $2k-1$ signed messages on the blockchain ensures that Bob gets X and Alice gets Y, Because the untrustworthy parties never have more than $k-1$ shares of either, it follows that Bob gets X if and only if Alice gets Y: fair exchange.

An alternative tactic for a corrupt party is not to do its job of posting a message when it has one share of each of X and Y. If they all cooperate on this then we will need $N \geq 3k-2$ to ensure that if Alice and Bob give all their shares to them then at least $2k-1$ will write that they have one of each.

We can therefore state this as a condition that guarantees that our algorithm will work.

Note that the use of threshold cryptography achieves the confidentiality goal that the intermediaries do not learn X and Y, provided that the communications around the network are appropriately encrypted.

The above protocol does not carry out the transfer when the $2k-1$ threshold is not reached, without expressing a limit on when it can happen. It would be normal to have a time limit on how long it can take, and ensure

that Alice, Bob, and as far as necessary the system achieve consensus on when an exchange has timed out. This is often one of the hardest parts of fair exchange, but fortunately the presence of a blockchain is very helpful here. What we need to do is have Alice and Bob agree a time limit up front, and replace the requirement that $2k-1$ messages appear at anytime, but the requirement that they appear in blocks timestamped before that limit T. Then as soon as a block with greater timestamp than T appears before $2k-1$ intermediate certificates, then all (at least trustworthy) participants can rely on the abandonment of the exchange protocol.

3.1 Alternative packets

Seemingly any mechanism for fair exchange without a single TTP has to divide up X amongst things to send to separate workers, since sending the whole thing would put it at risk.

The solution above of using threshold cryptography achieves this, and also achieves confidentiality of what is being sent. The latter may be an advantage but also a disadvantage, because there is no obvious way in which the workers can verify that X is what it claims to be.

This is potentially solvable at the expense of confidentiality. It we were to have A and B send any sort of tokens conveying X and Y, and ensure that these were not usable for value unless B or A have k distinct shares, then in terms of transferring value this would be identical to the one involving threshold cryptography.

For example, the tokens could be of the form $(A, B, X, i, k)_{sk(i)}$ where A is transferring X to B, and that this is distinct token index i with k required.

With tokens of this form, it will be possible for each of the workers W_i to verify the correctness relative to the contract of X and Y, and only sign the intermediate certificates when satisfied.

3.2 Alternative voting

The two schemes above both assume that all workers have equal influence. In other circumstances we might want to give them different levels of trust, or allow them to contribute to the process only by some amount of proof of work etc.

The first of these can be dealt with by giving different workers different numbers of shares of X and Y depending on how much they are trusted, or in the case where the shares are not from threshold cryptography simply

giving some shares more weight than others. (In either case presumably k would need to be adjusted to account for the total mass of shares.)

We can create a proof-of-work puzzle out of sending a single value such as a threshold share. For example if such a share is xyz where xy consists of effectively random bits, and x is of length r, then sending $(yz, hash(xyz))$ (where the hash is much longer than x) represents a puzzle which on average will take Bob 2^{r-1} hashes to compute.

Thus Alice can send Bob s shares, and he can compute as many as he likes subject to doing the necessary work. Alice might want to limit s to be less than some fraction of k.