Delay and escrow in the blockchain

(Draft)

A W Roscoe*, and Bangdao Chen
Chieftin Lab, Shenzhen and University College Oxford Blockchain Research Centre

Abstract

In this paper we show how to implement exact-time delay encryption in a trust environment like the blockchain, where we can be confident that some sort of majority of participants are trustworthy but not any individual one. In other words we give a protocol for generating $delay(x, t)$, a value which gives no significant information until time t, whereupon it can be decrypted to x by anyone. We highlight some applications of this construct, show how it can be extended to a more general form of escrow, and show how both of these constructs have interesting applications.

One of these is the creation of a fair and unbiassable source of random numbers in the blockchain: a *random oracle*. This in turn has many applications.

1 Introduction

Time-lock encryption [1, 2] was first described in the 1990's, generally with the implicit assumption that the delay would be long. $delay(x, t)$ is, to us, a value which can be decrypted by anyone at time t or beyond, but by no-one before this time. In [4, 5], the first author showed how it could be used to create protocols that are harder to attack and a mechanism for stochastic fair exchange. In both cases this is done by carefully denying participants information that might allow them to cheat, or cheat more effectively, if they had it too early.

In these applications there is no hard-and-fast schedule for the delayed information to become available. All the protocols need is that it does not

*Also Oxford University Department of Computer Science

become available before some t that represents a time limit for some protocol action being completed, and can be extracted by anyone some reasonable time after t. There is no requirement that the contents are available at precisely t. We will term this *lower bound delay*, and we expect this to be used most frequently in applications where the consequence of opening a delay is to complete a process without a strong timing requirement on its completion. This admits an implementation without a trusted third party (TTP): the party creating it applies a function with a publicly known inverse that takes a significant amount of sequential computation to compute; sufficient so that no-one can compute the inverse before t.

It is not obvious, however, how to implement *exact delay* (where anyone can obtain x immediately on t) without a TTP. With a TTP there are various options. In Section 3 we will show how it can be achieved. We then build a model of a blockchain system and show how it can both work on that and provide useful security for some trading applications and smart contracts. We then show how the same ideas can be used to create a blockchain generalised escrow system.

The applications we have in mind in this paper will have much shorter delays than those imagined in the early papers: perhaps seconds or minutes or the time it takes for blocks in a blockchain to become immutable. At any rate far too short for an advance in algorithms, hardware or cryptanalysis to make much of a difference between the time of encryption and the time of release.

In this paper we take it as given that all nodes actually know the time to a high degree of accuracy: they do not need to read it from an agreed central resource. However they can potentially do things they are not supposed to or fail to do things that they are, whether time-based or otherwise: so they might for example lie about the time. We will concentrate on ensuring that overall goals are met despite some of the nodes misbehaving. Of course there are also potential threats to a distributed protocol from attacks on the network combining them, but that is an orthogonal issue that the likely implementation using a blockchain helps to address, but not entirely.

In the next section we discuss how to implement exact delay in a blockchain and similar trust environments. We then look at some example applications of that before extending the idea to generalised escrow. Finally we show how delay encryption allows more secure implementation of the blockchain itself by giving it a fair choice mechanism.

2 Implementing exact delay

If we had a TTP (Trusted Third Party) Sam then exact delay could be implemented as follows (as reported in [4]). Sam is programmed to create a new key pair (pk_r, sk_r) for each time in a series $t_0, t_1, t_2, \ldots$. Well before time t_r, Sam signs a certificate announcing that pk_r is the key for time t_r. At time t_r (not before or after), Sam releases sk_r. This approach scales far better than getting Sam to actually hold onto the delayed values, because the amount of work Sam does is completely independent of the amount of material being delayed: no work by Sam is required for any individual delay.

Supposing for the sake of argument that the pk_r are generated and posted no more than a day before time t_r, we note it will be possible to revise things like key length and signature algorithm dynamically so that the system can remain confident of their security, but there is most unlikely to be any need to do this for a time t_r where pk_r has already been released.

Now Alice can create a delay encryption of X to any time t_r: she simply reads pk_r and then $delay(x, t_r)$ is $\{x\}_{pk_r}$ (where it might be desirable to add salt depending on the application). Clearly no-one other than Alice herself and Sam can obtain x beyond the appointed hour.

Of course if Sam were not trustworthy he could fail to deliver sk_r on schedule, release it early, or tell his friends the value early. It would not normally make sense to send a delay encryption built using Sam's keys to him, because he already has access by knowing the key. Thus Sam might be an agent who only implements this and similar services, or there might be a choice of TTPs so that an independent one can be chosen.

Note that there is limited need for the release of sk_r to be secure, since it can be checked against securely signed pk_r , though there may be pragmatic reasons for signing it or securely releasing a hash of it together with pk_r as this, as this may cut down the potential for attackers causing mischief.

Now we will study the situation without a conventional TTP, but with a collection of parties bearing resemblance to a blockchain. Specifically we assume we have a collection of nodes $W_1, \ldots, W_N$ where by common consent it can be safely assumed that any collection of k of them are not all corrupt, and where $N \geq 2 * k - 1$. So in particular more than half are assumed to be good, but we do not know which. We also assume that the nodes (as we assumed for blockchain nodes) all have an accurate clock, and that there is a workspace (which, when this is a blockchain implementation, might or might not be the blockchain itself) where nodes can post messages for the others to read, with this process being reasonably reliable.

Note first that if Alice has issued $delay(x, t)$ to someone before t, then

it might not be in her best interest for x to become visible at t– perhaps it records a bet which by t it is apparent she has lost – or indeed Alice might be offline at that time. It follows that Alice cannot be relied on to do her own releasing (which she could of course do). So we need to find a way to guarantee the release of x under the above assumptions.

One description of the blockchain is that it represents a trusted third party made up of many individually untrustworthy actors. However it is not the sort of TTP that is obviously usable for creating exact delay. We will form an abstraction of what can be trusted of it later. It is certainly improper for any individual node (and so potentially corrupt) to hold information that allows it to deduce the delayed value x directly (because it knows x) or indirectly (because it, for example, has access to both $\{x\}_k$ and k for some symmetric key k). We will assume that we have picked N actors within the blockchain that obey the trust model above.

Rather than have a single process creating key pairs, have our N selected W_i all individually generate them. We ask all of these N processes P_j to create a key pair (pk_{ji}, sk_{ji}) for each t_i, and individually to release the keys pk_{ji} and sk_{ji} on the same schedule and basis as outlined above. If we are using a blockchain we might expect the pk_{ji} to be placed on it. It is quite likely that the sk_{ji} need to be released in a prompter way since we do not want the delay release to get delayed by blockchain protocols. In any case they can be verified to match with the pk_{ji}, for example using hashes as indicated above. It is not necessary to use the same N parties W_j for each t_i, but we imagine it often be usually convenient to do so, or at least use a predictable schedule.

To create $delay(x, t_i)$ Alice now uses a threshold encryption scheme such as Shamir's [6] to deliver $M \leq N$ shares s_j of x such that any k of them reveal x but $k-1$ reveal nothing, where both Alice and in some sense the network itself are sure that there are not k corrupt members of the group, and at least $M-k$ will successfully deliver the necessary secret keys on time. She encrypts s_j with pk_{jr} (where available), and $delay(x, t_r)$ is just the combination of these $\{s_j\}_{pk_{jr}}$. It would be sensible for $delay(x, t_r)$ also to contain any necessary details of the encryption, the value t_r and a hash of the combination of x and all of these.

As a protection of the integrity of the scheme it may be wise to salt x with an unguessable nonce and thus include the nonce in the hash. Naturally the *delay* value, like any other cryptographic value, may be signed by the originator depending on context.

An untrustworthy participant can do one of the following to try to frustrate us:

- He can fail to produce pk_{jr}. But at least k do and furthermore Alice, in creating the *delay*, will not use pk_{ij} that have not been issued, going to other nodes' instead. Thus this is poor tactics for a corrupt node.
- Where he has released pk_{jr}, he can release sk_{jr} early or late. But at least k correct values do get released at t_r and the shares s_j deducible from the sk_{jr} released early tell us nothing.
- He can release wrong values for pk_{jr} of sk_{jr}. But the integrity of such a pair can be checked and has nothing to do with s_j.
- The last two are poor tactics, because they are both essentially public disclosures of untrustworthiness. A more dangerous misbehaviour is if it discloses sk_{jr} to selected parties covertly. However the fact that there cannot be as many sk_{jr} thus released for any t_r as needed means than no-one can obtain x early.

It follows that Bob (and everybody else who has $delay(x, t_r)$) can get k correct shares and deduce x, but that no-one can access x through this value before that. (It is assumed that the W_i are not themselves sent $delay(x, t_i)$ or that any good one who is is trustworthy enough not to conspire with $k-1$ bad ones.)

From the point of view of network security the main issue is ensuring that the sk_{ij} are promptly visible to all nodes once released. We imagine that this may include a degree of propagation or duplication. The communication and availability requirements our algorithms need are close to those required for blockchains.

We would expect Alice to follow a protocol for choosing $M \leq N$ and k, and where $M < N$ the nodes whose pk_{jr} are used. This might well be laid down for a particular application of *delay*. After all, Alice is not the only person to depend on the reliability of this *delay* value, and we would not want her picking only her special friends who might be partisan. We will discuss this later.

2.1 Blockchain assumptions and applications

The blockchain is, at the time of writing, widely touted as a solution to many problems in distributed data storage, asset registers, and transaction execution.

There are a number of different views of what a blockchain (or distributed ledger) is, what can be assumed of it, and how it should be used.

For us it is the following:

- A database with a collection U of users, of whom a subgroup M are "miners", namely those who build and maintain the chain. Some users can be tied to real-life entities, and some are anonymous pseudonyms.
- Anyone can write into the database. They have a choice of whether to sign such items or not.
- The miners decide which items succeed in being written by a consensus mechanism. They only have the right to reject a write if accepting it would violate a consistency rule of the blockchain (e.g. a double spending transaction). They use some consensus mechanism to achieve this.
- The miners create blocks of writes which are issued in a strict sequence, which is enforced by each non-initial block including a cryptographic hash of its immediate predecessor. The blocks are internally authenticated by hashing (Merkle Trees).
- They have a time-stamping mechanism that assigns times to items in blocks such that all times in a successor block are greater than all in its predecessor.
- Depending mainly on whether this is a public (i.e. anyone can mine) or private (mining is restricted to relatively few authorised parties) blockchain, there is the possibility that an issued block can be voted out of existence, so that history can change.

In a typical private blockchain it will generally be easy to assert the trust model assumed in the last section. Here anything less than perfect trustworthiness is assumed to be unusual, though not impossible. However in a typical public blockchain model, where trustworthiness is not assumed and correct mining behaviour arises primarily because of the carefully constructed incentive models, it causes problems unless there are a number of nodes who can probably and independently be trusted for separate reasons. We think it is a good idea, as proposed in [3] to motivate essentially trustworthy *mainstream* players to participate, to increase the general level of trust. Preferably they should do so publicly, not only so that they can be identified as trustworthy but also to make them more so as they will especially not want to be seen to do wrong. We imagine that the W_i chosen will generally be picked from such players if it can be agreed who they are.

Certainly, in private and mainstream blockstreams, we see no particular reason to identify the workers W_i chosen for the escrow task above with the

self-selecting miners. We will have an incentive mechanism by which self-interest will directly force the latter to behave themselves. With the former some degree of selection is possible and the incentives possibly more subtle, though only overtly detectable misbehaviour can be penalised.

Blockchains generally represent assets as unspent transactions: there is a transaction transferring some money, shares, land etc. to Alice, and she has not spent it yet. Transactions between anonymous identities are effectively anonymous: ownership comes down to knowledge of some key. So although everyone using a blockchain can see what transactions have happened on it, the fact that identities can be concealed, together with other information (such as what is being transferred) that is not essential to the ledger can be concealed.

So in particular all details of a transaction that are not required to be present simply for the blockchain to function can be delay encrypted.

Many stock exchanges and other services will require much greater transparency than this, meaning that things like the beneficial owners (before and after a transaction) may need to be recorded on a transaction. In current exchanges such information may be included but be restricted to certain parties, or only be made available (say) 30 minutes after the transaction. The first author was asked by a stockbroking firm how such things could be made consistent with a blockchain where everything was public. The answers seemed obvious: use encryption where the subsequent access did not increase with the passage of time, and exact delay encryption (possibly coupled with ordinary encryption) where is did. However he did not then know how to implement exact delay encryption without a TTP, so in a strong sense that conversation inspired the present paper.

The motivation for keeping transaction details secret is to keep the trading activity of some investor or broker secret so that others cannot make use of the information in deciding their own activity.

Such encryption can conceal who a transaction is between, but cannot conceal the fact that trading in some security (for example) is happening. It is, however, possible for anyone who owns something to transfer it between two identities he owns, and until the delayed information is in the open it will look exactly the same as a real transaction. Thus real trades can, to some extent at least, be camouflaged.

A smart contract is a piece of code representing a contract between two or more parties. It is placed on the blockchain, determined when it is to be executed and then carries out transactions. Because the blockchain can be read by all its users, anyone can examine the code. This means

- The conditions for executing the contract, and the nature of the trade it will execute, are public as soon as the code appears in the blockchain.
- Other parties are free to look for weaknesses in the contract, which might either be caused by a trader making an error or judgement or introducing a bug into the smart contract. Others can discover and contemplate such errors at their leisure and affect the market or take up positions which exploit the weakness: most likely to their own advantage and damaging one or more parties involved in it. The best known actual security flaw in a smart contract, the DAO attack [7] fell into this category.

Where the contract simply states the time at which it will be executed, it is straightforward to protect against this sort of problem: simply delay-encrypt the code of the contract until that time. The nodes with responsibility to execute it then decrypt it and run it at such a t.

Exact delay encryption is clearly also of great use in distributed sealed-bid auction and tendering protocols: bids must be sealed by delay ecryption until the time (after the end of bidding) when they are opened. This is effectively an anti-corruption measure. Who they open to (maybe everyone, all bidders, or an auctioneer) after auction closes is a matter for the designer.

It might also be used in e-voting protocols to prevent anyone from counting votes until the polls have closed.

3 Implementation

In any implementation we need to assess the threat model to decide how many parties need to generate keys for each time, and what number of them can be considered trustworthy.

Is there any group of nodes that are considered more trustworthy: perhaps the miners in a private blockchain environment? If so should generating key pairs be limited to them?

In environments like a public blockchain, what motivation do we need to provide for participants to perform their role and do so in a trustworthy]fashion. We imagine that the reward will take a similar form to that for mining, and that a node will be severely penalised for doing something wrong unless (in the case of failure to post keys) it has a good excuse. Noting that a node passing keys early to its friends is not necessarily spottable, we can institute a mechanism whereby any node that demonstrates knowledge of another node's secret key early can claim a large penalty from the

provider as well as disgracing it.

By and large the penalties for failing in one's duty might be so large that the likelihood of any not delivering keys as required is very small.

It will also have to be decided what granularity time will have: will key pairs be issued per second, minute, hour or day?

If this granularity is smaller than the rate at which a blockchain delivers blocks, then we cannot rely on the blockchain as the mechanism for broadcasting secret keys, though there is no problem with recording the public keys, since they can be posted in groups before they are needed. Note that a secret key can always be verified relative to an already-posted public one. We will discuss this issue again later.

There are various potential mechanisms here, depending on circumstances. It may well be that some external agency is accepted as a reliable model of time and is used for the timestamping of blocks. Possibly the release and availability of secret keys can be judged by a collective or fault-tolerant mechanism judged relative to this.

It must be clear, however, that the time intervals between keys which must be verifiably released at regular intervals must be several times greater than the latency of the network connecting the nodes. And the times at which individual nodes have released information will vary a little from the target t depending on the transmission of released keys.

If the only scource of trading information is the underlying blockchain then there may be no reason to have a higher rate of release of secret keys. However in general we must expect that information is probably coming from more rapid data streams. As discussed later, we do not normally expect the blockchain to be the source of timing information in *delay* as all nodes can be expected to own an accurate clock.

4 Generalised escrow

One can imagine a generalised *delay* operator that releases its contents under more general circumstances than the arrival of a particular time. If r is condition based on time t and features of the state s that are

- Observable deterministically everywhere with the same result.
- Once true remain true: a change of state cannot make such a condition false when it was true. Therefore in essence they are equivalent to something of the form "There has been a past state such that P". We will examine this concept further below.

then it makes sense to escrow information x so that it is released when r is true. The conditions above make it unambiguous when x is to be released, even when different nodes may make independent assessments at silightly different times.

We can thus imagine a generalised form of *delay* that we can write $escrow(x, r)$. This can be implemented in exactly the same way as *delay* except that it is not reasonable to expect nodes to create key-pairs tied to arbitrary conditions without prompting. It follows that anyone creating $escrow(x, r)$ will need to obtain the keys from enough parties and tie them to r so that it can create the encrypted shares of x that are needed. There will thus need to be a marketplace of keys which can be obtained from other parties who are prepared to release a public key tied to r (by signature) and monitor r to determine when to release the secret key.

We can imagine that there may be keys available for sale by potential (identifiable) W_i, and that a user will group them, placing such a group on the blockchain in advance of them being needed with an anti-double-spending mechanism being used to prevent any being used in different groups. When used the r will be attached to the entry, for example at a location specified in the blockchain with the entry for the group.

Such keys can be reused if multiple xs are escrowed by the same r.

Examples of r are

- $t \geq t_0$ (giving the equivalent of $delay(\cdot, t_0)$)
- Company X has breached condition p (determiniable from observable information) at some previous time.
- A legal warrant for the release of x has been placed on the blockchain.
- X has committed a demonstrated crime on the blockchain.
- The price of shares in X has exceeded £5
- The price of shares in X was greater than £5 on 20 September 2017.

The information required to evaluate such r should be stored in a form where they can reliably be computed: the same timed information should be available to all. Of course information in the blockchain automatically has this property.

Earlier in this paper we imagined that *delay* could be used to secure smart contracts that are triggered by a specific time. We can now generalise this to ones triggered by some computed condition r on the blockchain.

4.1 Trigger conditions on escrow

When considering delay and escrow we need to understand the nature of time. For delay as discussed above we assumed that the time required by the nodes involved is real time as measured by essentially agreeing clocks they hold.

However when we consider escrow governed by conditions intended to be determined by the blockchain, plus time, it only makes sense to consider time as measured by the blockchain: there must be a point in history as recorded on the blockchain, including its timestamps, at which the condition is true.

Potentially blockchain states are undone because blocks are replaced. If escrowed data has been released on account of the now-dead block, it is not really believable that it can be un-released when the block disappears. Therefore, without special arguments, a node N should only release a secret key sk_i counterpart to a pk_i bound to condition r when r is true in a block that N knows to be immutable. A fundamental property we assume of blockchains is that once node N knows block B is immutable then eventually all will. In this case it follows that if N is trustworthy and releases its sk_i, then all other trustworthy nodes with secret keys bound to do so eventually.

It follows from this that in a public blockchain where blocks do not become immutable until some time after they are created, escrow conditions must, either explicitly or implicitly, state that $escrow(x, r)$ only opens x when r has been true for sufficiently long.

The nature of the conditions we have assumed means that if there is a point in the settled blockchain history when an r is true, it will remain true afterwards.

We conclude that the nature of time in *delay* and *escrow* is different. In the former time means ordinary time, and release will be prompt. In the second time means logical time measured by the immutable history of the blockchain, and release need not be that prompt, though it is inevitable. How prompt it is will depend on the characteristics of the blockchain where everything is happening.

Returning for a moment to the subject of securing smart contracts, it is clear that any smart contract can now be separated into the condition that triggers it (which must be public) and escrowed code that it performed on this event. One of the actions of such a contract may be to perform another smart contract which is similarly delayed: our framework supports arbitrary nesting of this sort.

5 Second approach to escrow

With *delay*, the use of pairs of public and secret keys for each time point means that many nodes can delay to the same time without creating multiple times. With general escrow, this multiple use is not so likely. An alternate approach to escrow is for the node A creating $escrow(x, r)$ to create n threshold shares s_i of some key k such that '$\{x\}_k$ is the core of the escrow value, and send the pairs (s_i, r) encrypted to the W_i. W_i now has the duty to release s_i when r is true.

It would not normally make sense to send shares of x itself, as opposed to a key k encrypting x, to the W_i, as revelation of the s_i would then tell x to everyone, not just those who are privy to the escrow value.

This second approach avoids the work and potential quantum vulnerability of the approach using public key cryptography, but means that the W_i need to be slightly more actively involved. It also means that no reuse is possible.

Of course the quantum vulnerability of a given public key algorithm is best judged on an evolving basis and will be related to the length of time a key may have to last. With delay encryption, we have the opportunity to bound this; with general escrow, this may be impossible. As remarked earlier, we have the opportunity to update the algorithm and/or key length as time progresses.

6 Delay as an aid to blockchain security: an unbiassable random oracle

Cryptographic signature makes it possible for one blockchain node to recognise the origin of messages from others. Where a node is trustworthy we may be able to guarantee that the messages it is obliged to send will be sent, and we can strive to make message transmission reliable. However either because of an innocent power or communication outage or a loss of service attack on nodes or intrastructure, a communication might not get through despite the best intentions of the sender.

On the other hand sending a message early might well cause undesired information leakage or insecurity. Consider the following example: it is highly desirable, in a blockchain, to have an agreed source of random numbers associated with a series of times, for example the times at which it is intended that blocks are produced. Given that no single node is trusted, it seems appropriate that every node, or at least sufficient nodes that there will

be at least several trustworthy amongst them, contribute to each of them. (Of course if there were a single universally trusted and availabie source of genuinely random numbers this would not be necessary, but that seems unlikely.)

Suppose that each of a sufficient collection of nodes has committed a value for a given time t_n by writing a hash of it in the blockchain, and such values are now immutable. Then some of these will be trustworthy and random. If, at time t_n, all these V_{nm} are released by the nodes N_m so that they can be checked against the committed ones, then as can treat the bitwise exclusive or of them as a reliable *random oracle*, in other words a source of random numbers that cannot be controlled by a small group of parties and thus each node can regard it as completely unbiassed. For necessarily the trustworthy nodes will have created their own values randomly, uniformly and independently of all the others, so V_n, the XOR of them, is random string of independent bits whatever villainy the others have engaged in.

Unfortunately we cannot rely on either an individual trustworthy or untrustworthy node to actually deliver their V_{nm} at the right time. The trustworthy ones (the good guys) might get blocked for the reasons above, and the untrustworthy ones for the same or through sheer villainy.

There are two choices that obviously result: to accept the values that are revealed (noting the danger that different parties might not see the same ones) or abandon the calculation in the case where they are not all available.

The fact that trustworthy parties may innocently fail to deliver their obligations makes it unreasonable to penalise non-deliverers severely. Given this, abandoning the calculation when someone fails to deliver is not practical since this might always be the case. Even if we could ensure that all parties see the same set of V_{nm}, there is also a distinct danger in only using the V_{nm} that do appear. This is that villains, who may be colluding with each other, may wait until the good guys have released their shares and then, *in full knowledge or the effect would have on the final result*, decide which of their own V_{nm} to release to optimise the resulting V_n for them: this is not acceptable since we do not want them to be able to do this.

While in some circumstances one might be able to argue that the above attack is not feasible, it would be better if it were not there at all.

It can be eliminated by delay encryption. First we note that while non-delivery of obligations is difficult to penalise because it is hard to lay the blame, no such restraint is necessary when a node delivers a signed contribution that is demonstrably wrong by the underlying protocol. In human terms it is like the difference between a sin of omission and one of commission.

Suppose that each node N_m has had to commit via hashing to its V_{nm},

and has the obligation (either at the same time or a later one) to place the delay encryption of V_{mn} on the blockchain before some time $t'_n < t_n$, where this will be immutable by t_n. It is now agreed that only those V_{nm} which have both the hash commitment and delay present at time t'_n will count towards the calculation of V_n. We note

- By blockchain properties all will agree on which count, and at t'_n no node can have any idea what any of the V_{nm} provided by another trustworthy party is. It follows that provided that at least two good guys do achieve this, no-one can in any way control the final V_n. They can affect it: if a villain makes the choice to take part or not it is changing V_n, but from its own point of view it cannot change the distribution of possible outcomes.

- The delayed version of V_{nm} can be used to reveal this contribution in the event that N_m does not do so at t_n.

- When V_{nm} is revealed directly or through unwinding the delay, the result should be checked against the hash commitment. This is helpful for the delayed version as the nature of threshold encryption is that a supposed example of this decoded from different shares might produce different answers if it was not genuine. Thus we cannot tell that any collection of k (the threshold) values is not drawn from a true threshold encryption.

- Analysing this, we realise that our new protocol does not guarantee to deliver a random oracle, but instead guarantees to deliver either a value V_n that is random and independent of any party's choice, or the demonstration that one or more of the nodes has committed a sin of commission.

- Of course much communication at time t_n, namely of the time-release keys s_n, is necessary to implement this. However the fault tolerant nature of delay encryption gives considerable latitude for individual communications to fail.

Such a random oracle can be enormously useful in a blockchain: it provides a mechanism for *the blockchain itself* to make choices, for example in mining or performing some desirable role. (In many informal discussion of blockchain protocols and algorithms, one reads about the blockchain choosing this or that, with the implication being that the choice is unbiassed and secure. The random oracle gives a secure implementation of this.)

This example shows that we can use delay encryption to implement protocols that require nodes to deliver values *at a specific time* that have hitherto been secret, even in a blockchain environment where nodes and communication do not have to be reliable. It prevents non-compliant nodes hiding behind supposed communications failures. Mischief will be plain for all to see, and the villains can be named, shamed and punished.

Looked at like this, the use of delay encryption in this way within the blockchain is enormously similar to its use in [4].

7 Conclusions

We have demonstrated that threshold cryptography is the key to implementing exact delay encryption in an environment where the majority of players can be assumed trustworthy. We need to be able to pick a selection of workers, for example in a blockchain, whom we can trust to satisfy the condition that less than k will not perform their roles both accurately and with integrity.

The random oracle that we described in the last section may well be involved in picking the selection of nodes that implement delay encryption. Since the random oracle depends heavily on delay encryption, this seems to be circular. However it is not since the oracle that picked the nodes to implement delay to moment t_n depended only on the correctness and security of delay to strictly earlier times. Thus, like much of the coherence of the blockchain, this becomes an inductive security argument.

Delay may benefit from, but does not require, a blockchain to support it. The notions of escrow we have provided really require a blockchain because we need that nodes will agree on the sequence of states so that they do not differ on the truth or falsity of the trigger conditions.

The essential feature of both the *delay* and blockchain-based *escrow* we have defined it that all trustworthy nodes will agree on when and whether to release their own share of the value being released. Note that usually we want that a delay or escrow value remains a distinct term Y which is sent between players and that in order to get at the unencrypted value x that is being delayed or escrowed any node will both need Y and whatever us released by the W_i at the point of release. This is crucial to enable players to be selective about to whom they send, for example, a delay encryption.

References

[1] Time-Lock Encryption. http://www.gwern.net/Self-decrypting. 2011.

[2] R.L. Rivest, A. Shamir and D.A. Wagner. Time-lock puzzles and timed-release crypto. 1996. http://bitsavers.trailing-edge.com/pdf/mit/lcs/tr/MIT-LCS-TR-684.pdf

[3] A.W. Roscoe and Bangdao Chen *The greening of blockchain mining*, Available from www.tbtl.com

[4] A.W. Roscoe, Detecting failed attacks on human-interactive security protocols. In Cambridge International Workshop on Security Protocols (pp. 181-197). Springer, Cham., 2016

[5] A.W. Roscoe and P.Y.A. Ryan. Auditable PAKEs: approaching fair exchange without a TTP. Cambridge International Workshop on Security Protocols. Springer, Cham, 2017.

[6] A. Shamir, How to share a secret. Communications of the ACM 22.11 (1979): 612-613.

[7] Mehar, Muhammad Izhar, Charles Louis Shier, Alana Giambattista, Elgar Gong, Gabrielle Fletcher, Ryan Sanayhie, Henry M. Kim, and Marek Laskowski. Understanding a revolutionary and flawed grand experiment in blockchain: the DAO attack. Journal of Cases on Information Technology (JCIT) 21.1 (2019): 19-32.