Delay and escrow in the blockchain (draft)

A W Roscoe*Chieftin Lab, Shenzhen

July 24, 2018

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 and show how it can be extended to a more general form of escrow.

1 Introduction

Time-lock encryption was first described in the 1990's, generally the implicit assumption that the delay would be long. $delay(x, t)$ is a value which can be decrypted by anyone at time t or beyond, but by no-one before this time. In [**?**, **?**], the author showed how it could be used to create protocols and a mechanism for stochastic fair exchange by carefully denying participants information that might allow them to cheat, or cheat more effectively, if they had it to 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 become available before some t and can be extracted by anyone some reasonable time after t: we will term this *lower bound delay*. 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.

*Also University College Blockchain Research Centre

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 [**?**] 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.

2 Implementing exact delay

If we had a TTP Sam then exact delay could be implemented as follows. 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.

Now Alice can create a delay 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 anyone 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.

We imagine 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, 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 at t without trusting any single party.

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. What we can do, however, is exploit the same trust model assumed in the blockchain and give the participating processes additional capabilities which we assume are performed within the same trust model. The work in this paper is particularly suited to trsut model of private and *mainstream* hybrid blockchains. In each of these it is possible to identify parties who are motivated to work in a rustworthy manner for reasons other than because they cannot profit from not doing so.

Rather than have a single process creating key pairs, we assume that we can select N (may be all, may be not all the blockchain parties) participants

with the property that there is some k such that $2(k-1) < N$ and where no more than $k-1$ of the chosen participants is untrustworthy. 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 as outlined above.

To create $delay(x, t_i)$ Alice now uses a threshold encryption scheme such as Shamirs [**?**] to deliver N shares s_j of x such that any k of them reveal x but $k-1$ reveal nothing. She encrypts s_j with pk_{jr} (where available), and $delay(x, t_r)$ is just the combination of these $\{s_j\}_{pk_{jr}}$.

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.
- Where he has released pk_{jr}, he can release it 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.

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.

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". 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.

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 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 the author 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.

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.

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 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.

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 intervale must be several times greater than the latency of the network connecting the nodes.

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.

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".

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.

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.
- 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.

In situations where keys are released dependent on information stored in a blockchain, this approach (and indeed presumably any other approach to escrow) requires care if nodes depend on blocks that may later be deleted. For if enough nodes determine that condition r is true based on a history $h.b$ and release the keys associated with r, it requires a considerable leap of faith to believe that all who have heard these keys will forget them when told that b has been replaced by b'. In such circumstances it may therefore be necessary to tie the release of keys (or at least the determination of whether rs are true or not to the voting on blocks: if I base the computation of r on a given history, I must be prepared to vote for it. Furthermore the parameters chosen must then ensure that the given history has enough support to ensure its success.

This escrow is therefore much easier in blockchains where we can be confident of no branching history relevant to the determination or r.

Returning for a moment to the subject of securing smart contracts, it is clear that any smart contract can now be separated into the event that triggers it (which must be public) and escrowed code that it performed on this. 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 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 have also shown that this can be generalised provided we can have more elaborate key generation and an underlying data semantics like that of a typical private blockchain, which should also generalise to some public ones.