# Time-Lock Cryptography

Sending Messages to the Future

William Clarkson

william.clarkson@tufts.edu

Mentor: Ben Hescott

December 15, 2013

**Abstract**

In contrast with existing mainstream cryptographic methods, time-lock cryptography promises the ability to encrypt data which, by design, cannot be decrypted before a specific amount of time has passed. I will review the eminent work in the field of cryptography on the matter, and explore both the potential applications and the inherent limitations of the process. Finally, I will discuss my implementation of a simple time-lock algorithm and assess it's practicality.

## 1 Introduction

Cryptography is the fundamentally the study of methods by which sensitive human-readable information (the "plaintext") can be transformed into an obsfucated form (the "ciphertext") from which the original information can only be recovered by a trusted party, using secret information (the "key") which they possess but an attacker does not. As is convention in cryptography, I will use the names Alice and Bob to refer to the two parties who wish to secretly share information. Most methods of cryptography currently in use fall into one of two categories.

**Symmetric Cryptography** The same key is used to both encrypt and decrypt the message. Any party with the power to either encrypt or decrypt the message can also do the opposite. Since both Alice and Bob possess the same key, if either party is compromised, then an attacker could decrypt existing messages that Alice and Bob send each other, but also forge new messages to either party.

**Asymmetric Cryptography** Each party possesses a pair of keys—their "public key" and their "private key". The public key is widely available but the private key is never shared. To send a message to Bob, Alice encypts it with Bob's public key. Bob uses his private key to decrypt the message. Due to the structure of public-key algorithms, only the private key can be used to encrypt messages. Since Bob's private key is not shared with anyone, it is more difficult for an attacker to obtain than the shared key used in symmetric algorithms. To allow Alice and others to send him messages that only he can decrypt, Bob must only share his public key. In public-key cryptography, the roles of the public and private keys can also be reversed, so Alice can encrypt a message with her private key which can be decoded with her public key. While this does not protect the information from any attacker, the encrypted message could not have been generated without the use of Alice's private key, so in effect, Alice can be verified as the legitimate source of the information. These two modes of encryption can be used in tandem to allow Alice to send a message to Bob which only Bob can decrypt and which provides the guarantee that, if Alice's private key was not compromised, the information originated from Alice and was not tampered with [Tra11].

**Time-Lock Cryptography** The two methods presented above provide guarantees about *who* can encrypt and decrypt a message—with the proper keys, Alice and Bob can encrypt and decrypt messages with relative computational ease, and an attacker must brute-force the keys they are using, which will take prohibitively long if they have been chosen carefully. In contrast, time-lock cryptography has an inherently different goal: to provide a guarantee not about *who* can decrypt a message, but *when* they can decrypt it. One method of encrypting a time-locked message is to symetrically encrypt the message, release the ciphertext to the public, and rely on a trusted authority to release the key at a specified time and date. However, there are several key ways in which this could go wrong. The authority could somehow lose the key so the original information may never be revealed. The likelihood of this could be reduced by distributing the key to multiple trusted authorities. However, this increases the likelyhood of another negative outcome: if any one of the parties chooses to defect and release the key before the planned time, then the encrypted information becomes publicly available before the planned date. Depending on the contents of the information, this could have potentially disastrous consequences. This concern could be ameliorated somewhat by using an erasure code like `zfec` [WO12]. This would break the key into a number of partially redundant chunks. For example, a 4K file can be broken into a dozen chunks, any 4 of which can be used to reconstruct the original file. If each of the chunks of the key were distributed to a trusted authority, then several would have to defect in order for the key to be released early. While this improves the prospect, being forced to rely on a third party makes for a poor cryptographic scheme, analogous to trusting a messenger to deliver a secret message without reading it or allowing the message to be otherwise intercepted. In this paper, I will discuss the other known method for time-lock cryptography, which revolves around a proof-of-work scheme, meaning that a certain amount of computational time is required to decrypt a message.

## 2 To the Community

### 2.1 Background

In recent history, with our society's growing dependence on distributed computer systems, cryptography has become a field of utmost importance. In the age of Caesar, when messages were discretely dispatched by messenger to his generals, a simple transposition cipher may have provided sufficient security against the messenger or another party glancing at the message. However in our era, when an individual can transfer their wealth over the open channels of the Internet, where there could be numerous attackers observing traffic between a user's computer and their bank's server, much more sophisticated cryptography is required. The SSL/TLS protocol which enables secure online banking and transfer of other sensitive information relies on the existence of trusted certificate authorities. However, new technologies have recently emerged which allow secure online transactions without relying on a central authority, most notably Bitcoin. In short, it relies a public *block chain* which is a record of all confirmed transactions (which have each been signed with the private key of the user making them). Third parties perform a process called *mining* involves confirming transactions which have occurred and add them to the block chain, and which is incentivized by possible rewards for confirming transactions [Bit13]. The system is carefully designed so it is resilient to attempts to falsify transactions and account balances, all without relying on any centralized authority. The meteoric rise of Bitcoin indicates the growing relevance of distributed systems which do not rely on any centralized authority. In the age of nearly omnipotent government agencies like the NSA which could very well be manipulating even authorities we have previously taken as trustworthy, it is more important than ever to investigate more systems which are inherently distributed and cannot easily be compromised at one point with secret bureaucratic intervention. In light of this, I will focus on assessing several methods of time-lock cryptography which do not rely on any trusted authority to function. In the next section, I will discuss possible actions of such a system.

## 2.2 Applications

A number of interesting applications of time-lock cryptography are presented in [RSW]. Currently, online auctions rely on a middleman to record the bids of each participant and reveal the winner once the time is up. However, a distributed online auction system could allow each user to encrypt and submit their bid to the other users participating in the auction and each user could begin decrypting the bids of the other users to independently verify the winner. If each user encrypted their bid to withstand at least the time remaining in the auction when they submitted it, no participant could determine any other user's bid before the auction ended.

An individual may want to automate the payment of money on a regular schedule. With a digital currency like Bitcoin, one could encrypt a series of transactions with regularly increasing solve times so the recipient could, for example, decrypt one payment each month [RSW].

Consider a scenario where an individual has information which is currently incriminating, the release of which the individual wants to guarantee at a later date. A criminal may want to release a memoir about escapades for which they were not punished, but which must not be released before the statute of limitations has been exceeded. If they were to retain an unencrypted copy of the memoir, they would risk its discovery. With time-lock cryptography, however, the safe release of the information can be guaranteed. Mark Felt, famously known as the informant for the Watergate Scandal, could have encrypted a confession of his identity to be decrypted half a century later to ensure that the world could know his secret, in case he had passed away before being able to reveal it.

With the advent of Bitcoins and other currencies which are based on the posession of information rather than the possession of physical matter, a variety of possible schemes which leverage the ability to encrypt money or transactions are possible, such as the one above, suggested in [RSW].

## 3 Limitations

The simple method of constructing a timelock puzzle I will explain in the next section can be constructed in $\mathrm{O}(\log n)$ time, but will require $\mathrm{O}(n^2)$ time to solve. While this allows for the construction of puzzles in significantly less time than they will take to solve, it still means that it is not trivial to create puzzles which will be withstand decades or more of computation. Furthermore, it has been shown that this asymptotic gap is optimal, and cannot be improved upon [BMG09].

The fundamental problem with time-lock puzzles is that, without relying on a trusted third party, the only way that they can require the passage of a certain amount of time is to require the amount of sequential computation expected to correspond to the desired amount of wall clock time, since any algorithm which checks the actual clock time could easily be spoofed. Since this fundamentally relies on an expectation of the amount of real time required for a given amount of computation (i.e. clock speed), and the speed of different computers varies so widely at any given time, as well as constantly increasing, it is difficult to determine how difficult of a puzzle to create which will require no less than, but not significantly more than some given amount of real time.

The algorithm described below is inherently sequential, so more parallelized computation doesn't make it any more tractable to solve quickly. However, the algorithm does rely on the two prime factors of a large number remaining secret. It is currently very difficult to factor large numbers, but this is a parallelizable problem and could conceivably become much more possible in the future with the rise of quantum computing, and extremely fast single-purpose integrated circuitry, which has recently grown in popularity due to its ability to mine bitcoins significantly more quickly than with a traditional CPU. This effectively puts a cap on the longest effective time-lock puzzle, since an individual seeking to decrypt the message would find the prime factors of the large number used in the puzzle in order to solve it in $\mathrm{O}(\log n)$ time if it was quicker than solving it in the intended way. As the time required to find prime factors decreases, the usefulness of time-lock puzzles of this variety will decrease.

## 4 The Basic Algorithm

### 4.1 Creating a Puzzle

In this section I will describe the basic algorithm presented in [RSW]. We would like to encrypt a message $M$ such that it will require $T$ seconds to decrypt. We start by generating two large, randomly chosen primes, $p$ and $q$ and calculate:

$$n = pq$$

We then calculate Euler's totient function of this value:

$$\phi(n) = (p-1)(q-1)$$

Determine the fastest processor which the puzzle must withstand and find how many times per second, $S$, it can square a number modulo $n$. Then calculate:

$$t = TS$$

Generate a random key $K$ which is long enough that it will be difficult to brute force. Using a secure symmetric encryption algorithm $E$, encrypt the message using the key:

$$C_M = E(K, M)$$

Pick a random integer $a$ in the range $(1, n)$. We would like to encrypt K with:

$$C_K = K + a^{2^t} \pmod{n}$$

However, it would be expensive to directly calculate $a^{2^t}$, so we first calculate:

$$e = 2^t \pmod{\phi(n)}$$
$$b = a^e \pmod{n}$$

We can then much more efficiently calculate:

$$C_K = K + b \pmod{n}$$

The values $(n, a, t, C_K, C_M)$ are returned as the encrypted time-lock puzzle, and all other variables should be carefully destroyed.

## 4.2 Solving a Puzzle

Since the factors of $n$, $p$ and $q$ have been thrown away, the fastest way to solve the puzzle is currently to perform the repeated squaring operation:

$$b = a^{2^t} \pmod{n}$$

The original key can then be recovered:

$$K = C_K - b \pmod{n}$$

Then using the decryption algorithm $D$ which reverses the encryption algorithm $E$, used in the encryption process:

$$M = D(K, C_M)$$

Thus, the original message, $M$, has be recovered.

## 5 More Sophisticated Algorithms

Several other algorithms have been proposed which offer improved the ability to more quickly produce a puzzle resistant to more computation, and to more accurately specify the time required to solve the puzzle.

**Chained Hashing** Start by generating $n$ values, which we will call $a_1 \dots a_n$. Apply some function $f$ repeatedly to each value for some number of iterations $m$. This function can be repeated modulus squaring, or some other good hash function. These $n$ processes are independent and can be performed in parallel on $n$ CPUs. Call these results $b_1 \dots b_n$ where $b_i = f^m(a_i)$. Take the first result, $f^m(a_1)$, and use it to encrypt the next seed, $a_2$. More generally encrypt the initial value $a_i$ with the value $f^m(a_{i-1})$. Then, use the final value, $f^m(a_n)$ to encrypt the message, yielding the ciphertext $C_M$. The final time-lock puzzle is $(n, a_1, b_2 \dots b_n, f, C_M)$. To decrypt the puzzle, one must calculate $b_1 = f^m(a_1)$ and use it to decrypt $a_2$, so $b_2 = f^m(a_2)$ can be calculated, and so on. This allows a puzzle to be created with $n$ efforts of some specified amount of computation in parallel which requires the same $n$ efforts of computation to be performed *in series* [Gwe11]. Therefore, with access to 10 CPUs, one can, in a day, create a puzzle which will require 10 days to solve on any of the CPUs. This is for any general hash function. Using the repeated squaring method with the prime factor shortcut to determining the final value, one can additionally take advantage of the $O(\log n)$ vs. $O(n^2)$ ratio described in Section 3.

**Memory-Hard Functions** The repeated squaring algorithm described in Section 4.1 is constrained only by CPU speed the speed of accessing on-die memory like registers and primary cache. These speeds vary widely across machines. However, research has been on functions which by design require constant access to new data from main memory, which cannot be significantly sped up with a faster CPU or cache. While CPU speeds have been consistently improving, memory access speeds have been much more constant. In fact, the physical lower bound on memory access is $O(\sqrt{n})$, while the lower bound for speeds of circuits (assumed to be square) is $O(\log n)$. These lower bounds are enforced by the speed of light, and mean that CPU speed will always continue to outstrip memory performance [Per]. Therefore, algorithms which require frequent memory accesses can be used to more closely associate the computational work required to solve a time-lock puzzle with real elapsed time.

## 6 Description of Implementation

I implemented the simple repeated squaring algorithm described above in Ruby to assess it's real-world practicality[1]. I tested creating timelock puzzles and found that the time required to solve the puzzle is very clearly $O(n^2)$, as can be seen in Figure 1. The plot shows desired solve times (entered by the user) and the actual time required to solve the puzzle on the same machine. The $R^2$ value of 0.9996 indicates that, at least for short puzzles, the required time can be very accurately predicted, meaning that the user can specify precisely how long they want the puzzle to take to solve, provided the speed (squarings per

[1]My code can be found at `https://github.com/wclarkson/timelock`

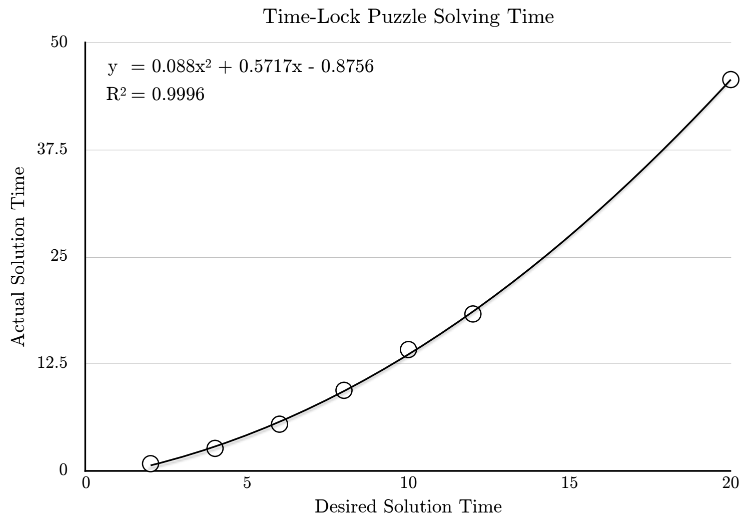

Figure 1: Illustration of expected O($n^2$) time to solve puzzle

second) the machine decrypting the message can perform.

## 7 Conclusion

In this paper, I gave an overview of the eminent research in the field on the subject of time-lock cryptography. It has many promising applications, but it is also important to understand the inherent restrictions on what can actually be accomplished. Finally, a working implementation of the algorithm of Rivest et al. was provided so the user can perform further testing of their own, and experiment with this novel cryptographic concept.

## References

[Bit13] Bitcoin.org. How does bitcoin work? 2013.

[BMG09] Boaz Barak and Mohammad Mahmoody-Ghidary. Merkle puzzles are optimal – an o(n2)-query attack on any key exchange from a random oracle. In *Proceedings of the 29th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '09, pages 374–390, Berlin, Heidelberg, 2009. Springer-Verlag.

[Gwe11] Gwern. Time-lock encryption. 2011.

[MMV11] Mohammad Mahmoody, Tal Moran, and Salil Vadhan. Time-lock puzzles in the random oracle model. In *Proceedings of the 31st Annual Conference on Advances in Cryptology*, CRYPTO'11, pages 39–50, Berlin, Heidelberg, 2011. Springer-Verlag.

[Per] Colin Percival. Stronger key derivation via sequential memory-hard functions.

[RSW] Ronald L. Rivest, Adi Shamir, and David A. Wagner. Time-lock puzzles and timed-release crypto. Technical Report Technical memo MIT/LCS/TR-684, MIT Laboratory for Computer Science. (Revision 3/10/96).

[Tra11] Wade Trappe. Public key cryptography: Rsa and lots of number theory. 2011.

[WO12] Zooko Wilcox-O'Hearn. zfec, 2012.