On Time-lock Puzzles and Timed-release Crypto

Jackson Blazensky

jackson@ucsb.edu

Timed-release Crypto Background

- Original paper: *Time-lock puzzles and timed-release Crypto*
- Rivest, Shamir (MIT, RSA) and Wagner (UC Berkeley), 1996
- Goal: "To encrypt a message so that it can not be decrypted by anyone, not even the sender, until a pre-determined amount of time has passed."
- Or more simply, "to send information into the future."
- The notion of time-lock puzzles (TLP) and timed-release crypto (TRC) were first described by Timothy May in the cypherpunks mailing list. May is a founder of the cypherpunks movement and the cypherpunks mailing list.

Timed-release Crypto Applications

- From RSW, and May:
- In an auction, a bidder wishes to seal his bid so that it can only be opened after the bidding period is closed.
- A homeowner wants to give his mortgage holder a series of encrypted mortgage payments. These might be encrypted digital cash with different decryption dates, so that one payment becomes decryptable (and thus usable by the bank) at the beginning of each successive month.
- An individual wants to encrypt his diaries so that they are only decryptable after fifty years.

Timed-release Crypto Applications

- From RSW, and May:
- A key-escrow scheme can be based on timed-release crypto, so that the government can get the message keys, but only after a fixed period (say one year).
- There are many more potential applications.
- From this author:
- Further intriguing potential use cases are enumerated later in this presentation. Stay tuned!

A Note

- We are used to thinking of encrypting data as keeping it safe *for all time* (practically speaking).
- This is to say, keeping it safe from a super-powerful adversary for so long that all the stars in the universe will burn out by the time they are finished brute-force guessing the secret key.
- This is an appealing notion, and we see it a lot in the Random Oracle Model and other models as proof of security, assuming we have cryptographic functions which are equivalent to Random Functions.

A Note

- By (dramatically) relaxing this standard way of thinking, we arrive at a weaker cryptography - by design. Such a cryptographic technique loses the ability to keep information secret forever, but gains in its place a new relationship with time and energy (work).
- Then let's assume we have a construction which can be broken in around a certain amount of time, or by a certain amount of work. Then the question is: What bounds can we guarantee in terms of time and work to unlock our secret?

Timed-release Crypto Approaches

- May's suggested approach requires trusted agents in its implementation.
- Inspired by May's idea, RSW succeeded in designing two different feasible implementations of TRC.
- These two approaches are considered to be the two *natural* approaches to delivering TRC.

Timed-release Crypto Approaches

- Approach 1: Use "time-lock puzzles" - computational problems that cannot be solved without running a computer continuously for at least a certain amount of time.
- Approach 2: Use trusted agents who promise not to reveal certain information until a specified date. (Along May's vision)
- We explore Approach 1 and leave Approach 2 for another lecture. (Both approaches are detailed in RSW '96 paper.)

Time-Lock Puzzles

- There are three obvious challenges for creating ideal TLP's:
- 1. The CPU time required to solve a problem depends on the amount and nature of the hardware used to solve the problem.
- 2. The CPU time required to solve a problem depends on the parallelizability of the computational problem being solved.
- 3. It is nontrivial to make "CPU time" and "real time" agree with precision.
- Let us start by presenting a naive approach which cannot deliver on these 3 criteria. Then we will have demonstrated a poor quality TLP which doesn't work.

"An unworkable approach" for TLPs

- Let M be our *message* which we wish to be locked for a period of time.
- Let S be the *speed* of a computer measured in decryptions per second.
- Then to encrypt M to be decryptable after T seconds, we choose a conventional cryptosystem CS with a key size of approximately $k = \lg(2ST)$ bits and encrypt M with a k-bit key.
- We save the cyphertext and throw away the key.
- By using exhaustive search of the key space, a computer will take about T seconds, on average, to find the key.

"An unworkable approach" for TLPs

- Actually, this method was first invented by Ralph Merkle. And the idea of these Merkle puzzles would lead to the idea and invention of public-key cryptography (and much more!).
- However this kind of puzzle construction will not work for our purposes of creating an ideal TLP. There are two main reasons.
- 1. For a conventional cipher a brute-force key-search is trivially parallelizable. This is sometimes called *embarassingly parallel*, and it means that N computers make the computation run N times faster.
- 2. The computation time estimate of T seconds is only an expected running time. The actual running time could be significantly larger or smaller, depending on the order in which the keys are searched.

RSW TLPs

- RSW directly address these problems in their proposal for a TLP which is designed to work much better than such a Merkle puzzle.
- To avoid the aforementioned pitfalls, the RSW stated goal is to "design time-lock puzzles that, to the greatest extent possible, are **intrinsically sequential** in nature, and can not be solved substantially faster with large investments in hardware."
- RSW TLPs aim for the following property: Joining computers to work together in parallel does not speed up finding the solution.
- "Solving the puzzle should be like having a baby: two women can't have a baby in 4.5 months."

RSW TLPs

- Since there is disparity between the speeds of individual computers, even at the level of gates, then RSW TLPs are *approximately controllable* in their timing. This means there may be situations which need more exact guarantees on precision timing of revealed information and thus one may do well to consider other kinds of TLPs if they exist or some other notion of TRC which more ensures precision. One such other form of TRC is the second approach not detailed in this presentation, but additionally investigated by RSW in the same paper.

RSW TLPs

- RSW TLPs are not automatically solved, but require dedicated sequential computing in order to solve. So if a puzzle is meant to be unlocked in 10 years, but a decrypter begins solving the puzzle 5 years after the puzzle is made available, then they may expect to solve the puzzle around 15 years after the puzzle was initially published.
- Another assumption built into the RSW TLP design: One must take into account the increasing speed of computation available in the markets over the time that the puzzle is to be solved. So we see a predictive element arise, and more so the longer the puzzle is intended to be locked. Therefore, RSW TLPs are *more* accurate and reliable the *less* the duration within which the puzzles are intended to be locked.

Creating a TLP

- RSW method is based on repeated squaring, and is an application of the *random-access* property of the Blum-Blum-Shub "x^2 *mod* n" pseudo-random number generator.
- Alice has a message M that she wants to encrypt with a time-lock puzzle for a period of T seconds.
- She generates
  1. a composite modulus $n = pq$ as the product of two large randomly chosen secret primes p and q.
- She also computes
  2. $\phi(n) = (p-1)(q-1)$

Creating a TLP

- She computes
  3. $t = TS$, where S is the number of squarings modulo n per second that can be performed by the solver.
- She generates a random key K for a conventional cryptosystem. And let us say that the encryption algorithm used in our conventional cryptosystem is E. Key K is long enough that searching for it is infeasible, even with the advances in computing power expected during the lifetime of the puzzle.
- She encrypts M with key K and encryption algorithm E, to obtain the ciphertext
  4. $C_M = E(K, M)$.

Creating a TLP

- She picks a random a modulo n (with $1 < a < n$), and encrypts K as 5. $C_K = K + a^{2^t} (mod\ n)$.
- To do so efficiently, Alice first computes 6. $e = 2^t\ (mod\ \phi(n))$,
- And then computes 7. $b = a^e\ (mod\ n)$.
- Finally, Alice produces as output the time-lock puzzle (n, a, t, C_K, C_M), and erases any other variables (e.g. p and q) created during her computation.
- Choosing p, q and a at random should yield the desired difficulty of the puzzle with overwhelming probability. However, picking a to be 2 should also be safe in practice. Over-optimization in the number-theoretical setup is considered "overkill" by RSW.

Solving a TLP

- By the design of the puzzle, searching for the key K directly is infeasible.
- Then the fastest known approach to solving the puzzle is to determine $b = a^{2^t} (mod\ n)$ somehow.
- Alice, in knowing $\phi(n)$, enables 2^t to be reduced efficiently to e, modulo $\phi(n)$, so that b can be computed efficiently by step 7, to create the puzzle.
- *But*, computing $\phi(n)$ from n is provably as hard as factoring n. So once Alice publishes the puzzle and throws away the key (the factors p and q) there appears to be no faster way of computing b than to start with a and perform t squarings sequentially - each time squaring the previous result.

Solving a TLP

- Alternatively one may choose to mount an attack, solving the puzzle by factoring n. But as long as p and q are large enough then factoring n becomes much harder than repeatedly squaring. Thus it is easiest and more time-and-cost-effective to work honestly to solve the puzzle as originally intended by the publisher.
- Repeated squaring seems to be an **intrinsically sequential** process - just as RSW set out to find when they defined the goal of their TLP design. There is no obvious way to parallelize this process to any significant degree.
- This is true even though a small amount of parallelization is possible to implement *within* each squaring.

Concluding RSW TLPs

- Ultimately we see that in RSW TLPs,
- "Having many computers is no better than having one. (But having one fast computer is better than one slow one.) The degree of variation in how long it might take to solve the puzzle depends on the variation in the speed of single computers, and not on one's total budget. Since the speed of hardware available to individual consumers is within a small constant factor of what is available to large intelligence organizations, the difference in time to solution is reasonably controllable."

Concluding RSW TLPs

- $O(log\ n)$ to create puzzle, $O(n^2)$ to solve the puzzle. This gap in complexity has apparently been shown to be optimal - it cannot be substantially improved upon.
- The number of t squarings required to solve the puzzle can be exactly controlled. Therefore we can create puzzles of any arbitrary level of difficulty.
- There is a simple and brilliant checking mechanism designed by Shamir that indicates whether or not a squaring modulo n operation made an error previously. This allows one to rewind to the point of mistake and continue to solve the puzzle correctly moving forward. Imagine all the potential years of wasted compute resources due to one mistake in a sequence!

22 Years Later

- RSW TLPs are still the only TLP candidate with any merit. They have been tested in the wild, and have withstood the test of Father Time. No significant breakthroughs have weakened the prospects or usefulness of these puzzles.
- In fact, many original RSW TLPs are still being solved right now, and most of them are behind schedule (meaning the puzzles are stronger than intended) because at the time, it was expected that Moore's law would continue at its steady pace of increase. But now we see that Moore's law is mostly continuing due to massive parallelization in the markets - the exact kind of computation intrinsically limited in the RSW TLP scenario! CPU speedups have not kept pace with their earlier rates.

Towards "Egalitarian Computing"...

- Can we narrow the gap between the capabilities of honest individual users and the capabilities of massively powerful, parallel, resource intensive adversaries?
- Hashcash (a DDOS countermeasure) - primitive "proof-of-work" meant, for example, to protect email users from spam. However, the utility of this idea in email is very limited. This is because the protection comes at such a high cost to every single honest user.
- Later this would be adopted into the Bitcoin p.o.w. mining algorithm.
- Some interesting candidate crypto-designs which aim to bring us closer to egalitarian computing:
- Memory-Hard Functions, Sequential Hashing, Guided-Tour Puzzles, Lattice-based Cryptography, and ...
- **RSW TLPs**. These puzzles are, in the opinion of this author, underestimated.

... In Public Cryptographic Networks of Scale

- Centralization risk, miner collusion, and pooling of mining resources are all huge problems in the emergent cryto-economic-networks of scale (Bitcoin, Ethereum, etc.)
- See https://arewedecentralizedyet.com
- Virtually all of the largest crypto-currency networks have 3 or less entities that, in the event of collusion, can rewrite transactional history, as they hold together greater than 51 percent of computing power on the network.

... In Public Cryptographic Networks of Scale

- The orginal vision (Satoshi's) was an egalitarian one: "one-cpu-one-vote". How could this space improve if pooling together mining resources was economically irrational?
- RSW TLPs may hold a key to truly decentralizing these public economic networks. This could take place in the form of a new proof-of-work algorithm designed as an RSW TLP.
- But this is a topic for further discussion in another lecture! Partial results have been made by this author (even with exciting contributions from none other than Ramanujan!), as it constitutes much of his research endeavors.
- Thank you for listening!