Time-Lock Puzzle with Examinable Evidence of Unlocking Time

(Transcript of Discussion)

Wenbo Mao

Good afternoon. I'll be sticking to the auditability theme. This is about a protocol which was proposed by Rivest, Shamir and Wagner. It's a timelock puzzle and to start with I will look at what is a timelock puzzle and what is its use, and then look at the RSW scheme, and then it will be an obvious requirement for auditability, to establish that the puzzle can really be solved within the stated time.

Before going into details I look at what the puzzle actually is. It is timed-release cryptography, which takes a very long time, or any specifiable length of time, to solve. Once it's solved you then know some bits of crypto. It is based on RSA, and now there is argument about how to name RSA, somebody says it's the alleged trademark of the cryptography used, so somebody else says it's *secret order*, like *discrete logarithm*, address the problem rather than the inventor's name.

Now the applications of time-release cryptography. Obviously there are several, say a bidder wants to seal a bid for a bidding period, another thing is sending messages to the future, a secret to be read in 50 years' time, and another thing is key escrow architecture. Key escrow is this thing where there is a requirement to escrow some keys so that they can be recovered, and the danger is vast scale intrusion. So with timed-release cryptography it will take some time to produce a key, although we mustn't waste a tremendous amount of time, but vast scale penetration becomes infeasible, becomes an individual criminal does not have the resources, so this is an example of a real application.

Now look at the RSW scheme. It is based on a secret order to an element. Suppose Alice has a secret to encrypt with a timelock puzzle for t units of time to solve. She generates two big primes p, q and multiplies them to obtain n, and then picks a random session key K and encrypts with this the message M using conventional key cryptography to get C_M. Then she encrypts the session key K using RSA, by adding a^e modulo n to give C_K. Here a is a random element and this exponent e is defined as $2^t \bmod \phi(n)$ where t is the number of timesteps needed to solve the puzzle. Since Alice generated p and q she can compute this e easily, whereas without knowing the factorization you cannot compute $\phi(n)$. Now C_M and a and C_K are published, so this triple becomes the timelock puzzle. So if we analyse it we know that to decrypt message M from C_M you need obviously the correct key, assume this, and to decrypt K from C_K you need to compute $a^e \bmod n$. Without knowing the factorization of n it seems that the only known way to compute a^e is by a repeated squaring of a, so that is t multiplications.

M. Roe et al. (Eds.): Security Protocols, LNCS 1796, pp. 98–102, 2000.

This procedure is intrinsically sequential, since there is no way to parallelise it. No matter how you compute, you may have any tool, it will actually still get the same number of operations, so it's sequential.

Michael Roe: There's some fine-grained parallelism in computing a square, because if I write the number out in a power series and then two of them are multiplied together, it looks like I can do in parallel at least some of those multiplies. We've still got the additions which I can't parallelise but I can do a lot of the multiplication, so I can gain a fair bit on squaring on a parallel machine. I think you're right in the sense that there is a limiting part through the middle of it that can't be parallelised but I can gain about $\log n$ by having parallel processes that work on different parts of it. Beyond that I don't get any speed up at all, so as long as t is much bigger than $\log n$ then, yes, intrinsically sequential.

Bruce Christianson: The argument is that the time is *eventually* linear in t.

Matt Blaze: It seems that it may be non parallelizable for one instance, but somebody who wants to parallelise a large number of these can certainly do a large number in parallel.

Reply: OK, but for one instance you cannot do it, so that's not an attack to this model.

Virgil Gligor: So we are treating here the problem of time release, I want to release, for example a key, at a certain time, and to make sure the bits are not opened earlier than a certain date. Can I do that as a matter of the protocol as opposed to as a matter of encryption? There is a protocol which is presented at the annual Computer Security Applications Conference last December by Michiharu Kudo from IBM Tokyo Research Lab on a time release protocol, and this paper was purely based on classic public key cryptography, there was absolutely nothing new in there. So that gives the impression that you could do more or less what you do with this new sort of encryption scheme without this kind of encryption.

Reply: Oh, yes, I guess that's true, this is only one way.

Virgil Gligor: So perhaps it would be good to think about the differences between solving the problem with conventional cryptography as a matter of a protocol, or solving it with a new form of crypto.

Ross Anderson: There's a small point there, it's perhaps not prudent to say that the time is linear in t. There's a nice point that Serge Vaudenay made, that if you ask a really naive first year undergraduate how much longer it will take to search 128 bits of key compared with 64 bits of key, then you expect the student will say twice as long. Now given Moore's law, this is actually correct because if you're given $1M and 20 years to do key search, you do nothing at all for 19 years and six months and then you do the whole thing, and so your key search power is up exponentially and so it does take you twice as long to search 128 bits than 64. So you have to be mildly careful, I think, not to say

that it's ultimately linear in t, unless of course you put in another "two to the power of"[1].

Reply: OK, next slide. Before starting to solve the puzzle, you know that t is big and it takes an awful deal of time. Before doing that, you want to know that it is a good puzzle: can I really get a good message, or will I just waste the time t? And in this scheme there isn't any way to be sure, you just have to trust the puzzle-maker. And we know that that's a problem.

Actually, we know that release even part of the secret and we have the power to do factorization, one quarter of the length of n and we can factorize. So for RSA it's a big problem, not so straightforward.

So this work here is to look at how to add-on some way to gradually release the secret, without causing sudden factorization, until the secret is released enough. So this is adding a protocol to this scheme, because as I said it will allow Alice to prove her honesty in making the lock.

So again it is based on RSA, so Alice should start out, set up with $n = pq$ choosen so as to make –1 have positive Jacobi symbol. This is very easy to ensure using Blum integers, so that $p, q \equiv 3 \pmod 4$, and this is a standard way although not really recommended but it's not harmful. And as we know, most elements in RSA have a secret order, Alice can select an element g with hidden order T so that $g^T \equiv 1 \pmod n$. Alice publishes g and keeps this order T secret. And we know 1 and –1 both have positive Jacobi symbol and we know $g^T \bmod n = 1$ and so has positive Jacobi symbol and we know g has negative Jacobi symbol, therefore T must be even.

Now knowing p and q permits Alice to choose such a g by algorithm[2], and by setting p and q Alice can also choose the size of T to be large, but anyway this is the trick, how to set n.

Now Alice wants to give a proof, and prove the size of T without disclosing T and this can very efficiently done, it can be done with a protocol and it's basically a proof of the number of bits in T. Alice has published g and n and provided the number of bits in T is relatively small, less than 100 bits, it will be the case that T can be found in in the time complexity of the discrete log algorithm. It's well known that this can be done using $\sqrt{T}$ number of multiplications, so if we have a number we can set that up between T and $2T$ with that precise level of control, so such a proof is all that is needed to enforce the protocol for partial key escrow.

How to do this is based on extracting a small discrete log. So we now look at how it works, we know that T is even, and suppose U is the largest odd factor of T. If we set $h = g^U \bmod n$, we know that $1 = g^T \bmod n$, we also know that both +1 and –1 have positive Jacobi symbol, and g has negative Jacobi symbol, and U is odd, therefore h has negative Jacobi symbol, and h^2 has positive Jacobi

[1] If p, q are Blum integers then $\phi(\phi(n)) = 3\phi(n)/4$.

[2] We ensure that g is a quadratic residue mod$p$ but a non-residue mod$q$, whereas –1 is a quadratic non-residue modulo both p and q.

symbol; therefore h cannot equal +1 or –1, but we know[3] that therefore $h^2 \equiv 1 \pmod{n}$.

So we have $(h+1)(h-1) \equiv 0 \pmod{n}$. Since h is not +1 or –1, –1 is just $n-1$, so these factors $h+1, h-1$ must both be bigger than zero and less than n, and their product is a multiple of n, so actually p and q must divide one factor each. So the greatest common divisor of $h-1$ and n is p or q. Now this is your controllable way to prove that n can be factored within a specified time.

So finally I will open for discussion, for a proper setting size T, the square root $\sqrt{T}$ can be a reasonably big size for various applications. When T is bigger than 20 bits the square root $\sqrt{T}$ will be 2^{10}, which is about 1K steps, so you can choose a set-up of T which can be decrypted within minutes or which can be decrypted within 50 years.

I am going to assume any algorithm to extract small discrete logs also will not parallelise well, we know the optimum so far, we know — unless we have some breakthrough along the way — using M processors only achieves $\sqrt{M}$ speedup. This is not very efficient for parallelising. And finally I think this control is less fine than the rest, but this one gives you some control, that one gives you no control although it's fine. And parallelisation, this will give you some speedup. These are the properties. Questions please.

Gianpaulo Bella: This may be more a comment on the original puzzle than on your work, but this is the first time I've seen this. It strikes me that the variability in the actual solving time of the posed puzzle is going to be so wide that the time itself is not easily useable for practical purposes. I'll try to make myself clear. If you are the one setting the puzzle, then you have control over the number of multiplications that someone has to do. Now to translate this into clock time, one has to take into account the real time it takes to make the multiplications, which you assume here are the most expensive thing. Firstly it is not given that the solver's computer will be alike, so the solver might only solve it in say ten times this amount or a hundred times this amount. And secondly, Moore's law and all that stuff, this could actually shrink the time. So if we have this huge variability of several orders of magnitude from the actual time that this number of multiplications translates to, how can I build any sensible real world thing? "I want this to expire in 50 years, plus or minus three orders of magnitude," doesn't mean much.

Reply: Yes, I think your comment is valid, as long as there is such variation[4].

Ross Anderson: What's wrong with just saying, solve a puzzle that uses a block cipher such as, say, Serpent with 97 bits of the key unknown. Then you're relying only on Moore's Law and your assumptions surrounding that, you're not relying on assumptions about somebody finding the more neat parallelisable way of doing small discrete log.

[3] By induction, since –1 is a quadratic non-residue and both the non-trivial square roots of +1 have negative Jacobi symbol.

[4] It would be OK on a physically sealed box like a time capsule, that could not be dynamically re-programmed.

Gianpaulo Bella: I think this is not viable because this is something that could be easily parallelised, it doesn't have the sequential properties.

But also there's a problem with the key, the amount of time to get the secret key is only probabilistically bounded.

James Malcolm: Is the expectation here that the puzzle will *have* to be solved, or is the puzzle there to be solved if something goes wrong?

Reply: This depends on applications, here I have assumed that the puzzle has to be solved.

Matt Blaze: Since multiple messages can be encrypted under this key, the architecture that might make sense for this would be: at various intervals, using some large public process in which many people are contributing bits in some way to ensure this is secret, we all agree on what this key will be in such a way that nobody actually knows what it is. And then there is a single key for which we make some estimate when it is due to be solved, and a large and very well funded effort at finding it. Now technology advances could make it happen sooner and problems could make it happen later if the effort declined (or funding did), but it's probably easier to estimate the advance in technology at the high end than at the low end.

Ross Anderson: There's a weird thing happening here, which is we live in an almost mediary era of computational history, where the best processors are the ubiquitous ones to very close approximation. The Cray doesn't buy you very much more than a desk processor. Now this wasn't the case twenty years ago and it may not be the case twenty years hence, once there are computing devices of single electron memory or whatever, so the advantage that the agency has over graduate students may open up again.

Matt Blaze: It may open up again. But I like the notion that you have a large public process that has many secrets that it's protecting, what the information-hiding workshop described as angry mob group analysis. And I think that there's an element of this, too, if society wants this secret sooner it can devote more resources to it but it may be expensive.

Ross Anderson: As people move from quadratic sieve to number field sieve, the computation relies more on the linear sieving than on the linear relations stage, so parallelising certainly becomes inherently less powerful.