Sending Message into a Definite Future: Non-Parallelizable Case

Wenbo Mao
Trusted E-Services Laboratory
HP Laboratories Bristol
HPL-2000-86
7th July, 2000*

E-mail: wm@hplb.hpl.hp.com

time-lock puzzle, zero-knowledge proof

We construct a proof of membership protocol that uses $\log_2 t$ steps to prove an element to have the structure $a^{2^t} \pmod n$ given public values n, a, t where n is the product of two large secret primes. Such a proof serves a concrete basis of trust for a time-lock puzzle scheme and its applications in timed-release cryptography. The achieved efficiency expressed in $\log_2 t$ (number of modulo exponentiation) manifests plainly that the proposed proof technique is practical in the applications of timed-release crypto problems.

* Internal Accession Date Only
Approved for External Publication

Sending Message into a Definite Future: Non-Parallelizable Case

Wenbo Mao
Hewlett-Packard Laboratories
Filton Road, Stoke Gifford
Bristol BS34 8QZ
United Kingdom
wm@hplb.hpl.hp.com

July 3, 2000

Abstract

We construct a proof of membership protocol that uses $\log_2 t$ steps to prove an element to have the structure $a^{2^t} (\mathrm{mod}\, n)$ given public values n, a, t where n is the product of two large secret primes. Such a proof serves a concrete basis of trust for a time-lock puzzle scheme and its applications in timed-release cryptography. The achieved efficiency expressed in $\log_2 t$ (number of modulo exponentiation) manifests plainly that the proposed proof technique is practical in the applications of timed-release crypto problems.

1 Introduction

Rivest et al proposed a time-lock puzzle scheme [4]. A time-lock puzzle is a timed-release crypto problem which is given along with a routine for finding the solution within a precisely specified number of computational steps. Based on the puzzle scheme, the MIT Laboratory for Computer Science has implemented "LCS35 Time Capsule Crypto-Puzzle" and started its solving

routine on 4th April 1999. It is expected that the solution the LCS35 Time Capsule Crypto-Puzzle will be found in 35 years from 1999, or on the 70 years from the inception of the MIT-LCS (see a detailed description in [5]).

The time-lock puzzle scheme of Rivest et al is based on an asymmetric property of an integer-factorization-based crypto algorithm. Let n be a composite integer of two large prime factors and let a, t be any non-secret integers such that $gcd(a, n) = 1$. Knowing the factorization of n a puzzle maker can construct a puzzle (which we denote by $P_a(t)$)

$$P_a(t) \stackrel{def}{=} a^{2^t} (\mathrm{mod}\, n) \tag{1}$$

at ease. The construction can be done via using Euler's phi-function $\phi(n)$ to compute first

$$e \stackrel{def}{=} 2^t (\mathrm{mod}\, \phi(n)), \tag{2}$$

and then

$$P_a(t) \stackrel{def}{=} a^e (\mathrm{mod}\, n). \tag{3}$$

These computations can be done at ease because they need $O(\log n)$ multiplications only where $\log n$ is a relatively small constant. This is the computational cost for the puzzle maker to construct a puzzle.

The computational cost for a puzzle solver will be very different from $\log n$. In the puzzle scheme of Rivest et al, the task of solving a puzzle is to find $P_a(t)$ using the given public numbers a, t, n. Without knowing the factorization of n, $\phi(n)$ is unknown (assume that n is adequately large so that factoring of n is infeasible) and thereby one cannot compute a short e using (2). It seems that the only known method to construct $P_a(t)$ (other than via factoring of n) is to perform repeated squaring mod n starting from a. The puzzle maker can properly construct n such that almost all $a < n$ has a large multiplicative order mod n, and 2 has a large multiplicative order mod $\phi(n)$ (here "large" means $\gg t$ for any t that a puzzle may use; these two requirements are easily satisfiable by choosing n properly). Clearly t such squarings will reach $P_a(t)$ from a. What is important to notice is that t squarings are also necessary since without knowing $\phi(n)$, 2^t is a number of $t+1$ binary bits to be used as the exponent in (3) (the space requirement is not needed since 2^t has a straightforward structure). What is more important to notice is that there exists no effective way to speed up the needed t squarings. Parallelization of many processors will not do a good job because of the

following two reasons. First, each step of squaring can only be performed on the result output from the immediate previous step and hence we cannot cut the big job of repeated t squarings mod n into many smaller sub-jobs to be processed in parallel. Second, parallelization of one squaring step cannot speedup the problem in a great deal because performing one squaring only takes a trivial amount of computational resource and parallelization of such a trivial operation will be penalized by communication delays among the processors.

The huge asymmetric costs between that of a small constant number of multiplications for the puzzle maker to arbitrarily tune t in the range from triviality to the difficulty of factoring n, and that of t non-shortcutable and non-parallelizable squarings for the puzzle solver to solve the puzzle forms the uniquely desirable property to underlie the time-lock puzzle scheme proposed by Rivest et al.

The LCS35 Time Capsule Crypto-Puzzle has been constructed with a good faith of correctness. However there exists no efficient method for a puzzle maker to prove the correct construction of a puzzle. A time-lock puzzle usually takes a great deal of length of time to solve (in the case of the LCS35 Time Capsule Crypto-Puzzle, $t = 79685186856218$ and the estimated time for reaching the solution is 35 years with the consideration of the advances of the microprocessor technology). A proof of the correctness of a puzzle is necessary in order to interest solvers. Imagine that if a puzzle is so constructed that a maker can demonstrate with a high degree of confidence that the solution is actually the factorization of n (see Section 2 for how an example of embodiment), then solvers will be excited to compete for being the first person who can (counter)sign an open cheque payable to the first signer which designates n as the public verification key for the signature. Obviously, in absence of a validity proof, few will be interested in risking being fooled by a cheating puzzle maker.

Mao proposed different time-lock puzzle scheme which requires a solver to extract the discrete logarithm of a given element where the target discrete logarithm has a precisely specified length which forms the evidence of the correct construction of the puzzle and this evidence can be demonstrated in an efficient zero-knowledge proof by the puzzle maker [2]. However, unlike the non-parallelizable property for finding the $P_a(t)$ that we have discussed above, the problem of extraction of a discrete-logarithm can be parallelized (e.g.,

using the parallelized Pollard's kangaroo algorithm [3] due to Van Oorschot and Wiener [6]). Therefore Mao's time-lock puzzle scheme suffers from a parallelization attack.

In this paper we will construct an efficient interactive protocol for proof of membership regarding the language

$$L_n \stackrel{def}{=} \{\, a^{2^t} (\mathrm{mod}\, n) \mid gcd(a, n) = 1,\ t < n \,\}.$$

This is the first protocol that proves the structure in $\log_2 t$ steps. It will also prove that the public modulus n can be factored (and hence can entitle the first signer with due credit) at the end of t steps of repeated squaring mod n.

2 Time-lock Puzzle with Provable Correctness

2.1 Puzzle Construction

Let Alice be a puzzle maker. She begins with constructing a composite $n = pq$ where p, q are two large primes. She should construct n to satisfy

$$\left(\frac{-1}{n}\right) = 1 \tag{4}$$

where $\left(\frac{x}{n}\right)$ denotes the Jacobi symbol of x mod n.

Alice then constructs a time-lock puzzle pair $(P_a(t), P_a(2t))$ via the calculations in (2) and (3) using values a, t. For $P_a(2t) \in L_n$, $P_a(2t-1)$ which we denote by g, is a square root mod n of $P_a(2t)$. This root is itself a quadratic residue since it can be reached by $t-1$ squarings mod n starting from $P_a(t)$. As a quadratic residue, g has the positive Jacobi symbol:

$$\left(\frac{g}{n}\right) = 1 \tag{5}$$

With the knowledge of n's factorization Alice can also construct another element h such that

$$\left(\frac{h}{n}\right) = -1 \tag{6}$$

and

$$h^2 \equiv P_a(2t) \,(\mathrm{mod}\, n). \tag{7}$$

Since g and h are two square roots of $P_a(2t)$ and have different Jacobi symbols, we know

$$gcd(\, g \pm h,\ n\,) > 1.$$

The factorization of n using g and h is guaranteed because $g \not\equiv \pm h \,(\mathrm{mod}\, n)$ due to the conditions in (4-6).

Thus, if Alice has constructed a correct time-lock puzzle pair $(P_a(t), P_a(2t)) \in L_n \times L_n$ and then discloses them along with n, a, t, h, the puzzle can be solved in the manner of factoring n upon discovery of $g = P_a(2t-1)$ after $t-1$ squarings mod n, starting from $P_a(t)$.

Note that because anybody can construct a quadratic residue mod n from any given element (i.e., anybody can form a pair of square root and quadratic residue), the disclosure of h as a square root of $P_a(2t)$ will not reduce the difficulty for factoring n in any sense.

2.2 A Building Block

A building block of the proposed scheme is a simple protocol for proving the squaring relation between two discrete logarithms to the same base. It is simplified from the previous protocols for reasoning about discrete logarithm equality [1]. The simplification improves the verification performance by doubling the speed up.

Let elements a, y, z satisfy

$$y \equiv a^x (\mathrm{mod}\, n), \quad z \equiv a^{x^2} (\mathrm{mod}\, n).$$

Protocol $\mathtt{SQ}(a, y, z)$ will prove that

$$(\log_a y)^2 \equiv \log_a z \,(\mathrm{mod}\, Ord_a),$$

where Ord_a is the multiplicative order of the element a.

SQ

Common input: a, y, z, n;

1. Alice picks a random number $k < \phi(n)$ and sends to Bob:
   $A \stackrel{def}{=} (ay)^k \pmod{n}$;
2. Bob picks a random challenge c and sends it to Alice;
3. Alice replies with the response $r \stackrel{def}{=} k + cx \pmod{\phi(n)}$;
4. Bob accepts if $(ay)^r \equiv A(yz)^c \pmod{n}$, or rejects otherwise.

Theorem 1 *The properties of* SQ.

Completeness *If* a, y, z *satisfy* $(\log_a(y))^2 \equiv \log_a(z) \pmod{Ord_a}$, *then Bob always accepts the proof.*

Soundness *If* $(\log_a(y))^2 \not\equiv \log_a(z) \pmod{Ord_a}$, *then Alice, even computationally unbounded, cannot convince Bob to accept a proof with probability greater than* $1/Ord_a$.

Zero-knowledge *Bob (may be dishonest but with a bounded computing power) does not learn any information aside from the validity of Alice's claim.*

Proof

Completeness Immediate from inspection of the protocol.

Soundness Alice's response has allowed Bob to generate $yz \pmod{n}$ from $ay \pmod{n}$. So there exists x such that

$$y \equiv a^x \pmod{n}$$

and

$$z \equiv y^x \pmod{n}$$

These imply

$$x \equiv \log_a(y) \equiv \log_y(z) \pmod{Ord_a}$$

Then by the logarithm property,

$$\log_a(z) \equiv \log_a(y) \log_y(z) \equiv x^2 \equiv (\log_a(y))^2 \pmod{Ord_a}.$$

If this congruence does not hold then in order to let Bob accept the proof, Alice must have prepared x to satisfy

$$x \equiv \frac{r - \log_{ay} A}{c} \pmod{Ord_a}.$$

Note that c is sent to Alice after she has committed $\log_{ay} A \pmod{Ord_a}$, therefore she will have at most 1 in Ord_a probability to have responded c with r correctly.

Zero-Knowledge We assume that Bob has a bounded computing power but may be dishonest. Then however the challenge c is chosen, he cannot control the distribution of $(ay)^r \pmod{n}$ with his bounded computing power. Such a proof view can be simulated perfectly. Consequently, Bob gains no information other than the validity of Alice's proof. □

The verification of SQ is very efficient, Bob only needs to compute two modulo exponentiation mod n.

Using a secure one-way hash function (let $H()$ denote a suitable secure one-way hash function), SQ can be turned into a non-interactive protocol. To generate a non-interactive proof, Alice performs the following.

Alice picks a random number $k < \phi(n)$. She then computes

$A \stackrel{def}{=} (ay)^k \pmod{n}$;

$c \stackrel{def}{=} H(a, y, z, A)$;

$r \stackrel{def}{=} k + cx \pmod{\phi(n)}$;

The non-interactive proof (certificate) consists of a, y, z, c, r, n and the description of $H()$.

To verify a non-interactive proof, Bob computes

$$A \stackrel{def}{=} (ay)^r (yz)^{-c} \pmod{n}.$$

He accepts the proof if

$$c = H(a, y, z, A)$$

or rejects it otherwise.

In our application to be described in a moment, Alice will always use SQ in the non-interactive version. For this reason we can view SQ as an algorithm and denote by

$$z := \mathtt{SQ}(a, y, n)$$

an instance of the acceptance run of the algorithm. By this denotation, we imply that in an acceptance run of the algorithm, z will be assigned with the following value:

$$z := y^{\log_a(y)} (\mathrm{mod}\, n).$$

2.3 Efficient Proof of Membership in L_n

Let Alice have constructed $P_a(t) \in L_n$ using an element a and t. Recall

$$P_a(t) \stackrel{def}{=} a^{2^t} (\mathrm{mod}\, n).$$

Since we can always express an even t as

$$t = 2 \times \frac{t}{2}$$

and an odd t as

$$t = 2 \times \frac{t-1}{2} + 1,$$

we can express 2^t as

$$2^t = \begin{cases} 2^{(2\times\frac{t}{2})} = (2^{\frac{t}{2}})^2 & \text{if } t \text{ is even} \\ 2^{(2\times\frac{t-1}{2}+1)} = 2 \times (2^{\frac{t-1}{2}})^2 & \text{if } t \text{ is odd} \end{cases}$$

Following these expressions, we can express $a^{2^t} (\mathrm{mod}\, n)$ as

$$a^{2^t} (\mathrm{mod}\, n) \equiv \begin{cases} a^{[2^{(t/2)}]^2} & \text{if } t \text{ is even} \\ (a^{\{2^{[(t-1)/2]}\}^2})^2 & \text{if } t \text{ is odd} \end{cases} \qquad (8)$$

In these expressions, $t/2$ and $(t-1)/2$ are computed in the integers, $2^{(t/2)}$ and $2^{[(t-1)/2]}$ computed in mod $\phi(n)$ (which can be done by Alice). These expressions translate into the following membership decision algorithm which will terminate within $\log_2 t$ steps and decide $P_a(t) \in L_n$.

In the algorithm below, := denotes the value-assigning operation, $\lfloor \log_2 t \rfloor$ denotes the integer part of $\log_2 t$, == denotes the test of equality, and $\div$ denotes division in the integers.

Membership$(t, a, P_a(t), n)$

$y := a^2 (\mathrm{mod}\, n)$;

$m := \lfloor \log_2 t \rfloor$;

if $(m == 0)$ **then** $z := a^2 (\mathrm{mod}\, n)$;

while $(m > 0)$ **do**

$z :=$ **SQ**(a, y, n);

if $(t \geq 2^m)$ **then** $t := t - 2^m$;

if $(t \div 2^{m-1} > 0)$ **then** $z := z^2$; **(* i.e., the odd case in (8) *)**

$m := m - 1$;

$y := z$;

accept if $(P_a(t) == z)$, **or reject otherwise.**

Clearly, a run of **Membership**$(t, a, P_a(t), n)$ will terminate upon completion of the $\lfloor \log_2 t \rfloor$ loops. Finally, $P_a(2t) \in L_n$ can be obtained by one more step of calling **SQ**:

$$P_a(2t) := \mathtt{SQ}(a, P_a(t), n).$$

As we have described in 2.1, the tuple $(t, P_a(t), P_a(2t), n, h)$ forms a correctly-formed time-lock puzzle, where h is a square root of $P_a(2t)$ with the negative Jacobi symbol. (Review 2.1 for the instruction for solving the puzzle.)

2.4 Performance

Because in each run of **SQ**, the verification job involves two exponentiations mod n and because **Membership** calls **SQ** $\lfloor \log_2 t \rfloor + 1$ times, the verification job for deciding $P_a(2t) \in L_n$ involves

$$2(\lfloor \log_2(t) \rfloor + 1)$$

modulo exponentiations.

In the LCS35 Time Capsule Crypto-Puzzle [5], $t = 79685186856218$ which is a 47-bit binary number. Thus the verification job that puzzle can be completed within 96 modulo exponentiations.

3 Conclusion

We have constructed an efficient proof scheme for providing the verification of the exact number of steps needed for solving a time-lock puzzle in the scheme of Rivest et al [4]. The efficiency expressed by $\log_2 t$ means an extremely low cost for providing such a proof, which is practical in the applications of timed-release crypto problems.

Acknowledgments

I would like to thank Professor Ronald Rivest for his encouragement on answering the challenge of the time-lock puzzle for the non-parallelizable case, and Kenny Paterson for interesting discussions of the topic on the train from Bruges to Brussels.

References

[1] Chaum, D. and Pedersen, T. P. Wallet databases with observers. Advances in Cryptology: Proceedings of CRYPTO 92 (E.F. Brickell, ed.), Lecture Notes in Computer Science Springer-Verlag, 740 (1993), pages 89–105.

[2] Mao, W. Send message into a definite future. Information and Communication Security: Proceedings of ICICS 99, (V. Varadharajan, Y. Mu, eds), Lecture Notes in Computer Science Springer-Verlag, 1726 (1999), pages 244-251.

[3] Pollard, J.M. Monte Carlo method for index computation (mod p), *Mth. Comp.*, Vol.32, No.143 (1978), pages 918–924.

[4] Rivest, R.L., Shamir, A. and Wagner, D.A. Time-lock puzzles and timed-release crypto. Manuscript. Available at (`http://theory.lcs.mit.edu/~rivest/RivestShamirWagner-timelock.ps`).

[5] Rivest, R.L. Description of the LCS35 Time Capsule Crypto-Puzzle `http://www.lcs.mit.edu/about/tcapintro041299`, April 4th, 1999.

[6] van Oorschot, P.C. and M.J.Wiener M.J. Parallel collision search with cryptanalytic applications. *J. of Cryptology*, Vol.12, No.1 (1999), pages 1–28.