LUC: A New Public Key System

Peter J. Smith[a] and Michael J. J. Lennon[b]

[a]LUC Partners, Auckland UniServices Ltd, The University of Auckland,
Private Bag 92019, Auckland, New Zealand.

[b]Department of Computer Science, The University of Auckland,
Private Bag 92019, Auckland, New Zealand.

Abstract
We describe public key cryptosystems and analyse the RSA cryptosystem, pointing out a weakness (already known) of the RSA system. We define Lucas functions and derive some of their properties. Then we introduce a public key system based on Lucas functions instead of exponentiation. The computational requirements of the new system are only a little greater than those for the RSA system, and we prove that the new system is cryptographically stronger than the RSA system. Finally, we present a Lucas function equivalent of the Diffie-Hellman key negotiation method.

Keyword Codes: E.3; K.4.2; K.6.5
Keywords: Data Encryption; Social Issues; Security and Protection

1. Public Key Encryption

Public-key encryption was first discussed by Diffie and Hellman [1] as a general principle. The new concept which they introduced was the use of *trapdoor* functions for cryptography. A trapdoor function is a computable function whose inverse can be computed in a reasonable amount of time only if a (small) amount of additional information is known. Both the function and the information needed to invert it are not dependent on the message being sent. As the trapdoor function is usually a numeric computation based on a number called a *key*, the initial computation and the inverse computation are usually expressed in terms of a *public key* process for the function computation and a *private key* process for the inverse computation. This concept finds its most secure application for cryptography in cases where the inversion process occurs at only one site. In this case, the private key (ie the information needed for the inversion process) can be kept in a highly secure manner. The public key process, however, can be widely publicised, so anyone can send an encrypted message to the possessor of the private key process. This message cannot be decrypted or changed in a non-detectable manner by a third party, since to do so, the private key process must be known.

As Diffie and Hellman pointed out, the trapdoor function technique can be used not only to encrypt messages to ensure privacy of communication, but also to sign them to ensure non-repudiability. In order for this to be possible, the trapdoor function involved must be mathematically bijective - in other words, the inverse of the inverse computation is the original function. In order to sign a document, the private key process is performed on the document, or possibly on a number (a checksum) derived from it by a publicly defined compression process. Any recipient of the message can use the matching public key process (previously publicised by the sender) to obtain the original document, or the checksum.

The main content of this paper is to describe a new public key system, the LUC system, which uses a trapdoor function based on Lucas functions. In the next section, we describe the best-known public key system, the RSA system, invented by Rivest, Shamir and Adelman (see [9]). The mathematics of this system is described in some detail, in order to point out the similarities and difference with the LUC system. In section 3, we describe the LUC public key system, including a mathematical proof that it does work. We also prove that, cryptographically, it is stronger than the RSA system, and requires an approximately equal computational effort.

Since writing the initial version of this paper, we have found that a public key system very similar to LUC has appeared in the literature on at least two previous occasions, but in neither case were details of a practical system given. In [6], Müller and Nöbauer describe a public key system using Dickson polynomials. However, they give no practical way of computing these encrypted messages. In a later paper [7] they show how to compute the encryptions in logarithmic time using a "square and multiply" technique; Williams' technique in [10] is an improved version of this algorithm. The results in Section 3 of that paper show that the Dickson polynomials they use are the same as the Lucas functions we use. In [11] Williams defines a rather complex system, using quadratic field extensions. The results in Section 2 in his paper show that his functions X_n are nearly the same as the Lucas polynomials we use $(2X_n(W_1,W_2) = V_n(2W_1,1))$, and the computation method he gives for X_n is based on his paper [10] which provides our computation method. However, his system has serious flaws (it was designed to obtain a specific result concerning factorisation, not to provide a practical cryptosystem) and our discovery that Lucas functions lead to a simple, strong, non multiplicative public key system is independent from his results.

2. The RSA Public Key System

Since Diffie and Hellman's paper, a number of possible trapdoor functions have been proposed. Of these, the best-known and most widely used are the ones used for the RSA encryption method, and for the El Gamal method. Both of these are based on raising the message (regarded as a number) to a power, modulo a large number.

The public key process used by the RSA method is defined by two numbers, e and N, which are used in the function:

$$f_{RSA}(M) = M^e \bmod N .$$

If M is a message, then $f_{RSA}(M)$ is the encrypted message, M'. In order for the inversion of f to be difficult to compute (ie to make f_{RSA} a trapdoor function), N is chosen to be the product of two large primes, p and q. For the private key process, a number d is needed such that

$$ed \equiv 1 \bmod (p-1)(q-1) .$$

If e is relatively prime to $(p-1)(q-1)$, then d will always exist and is easy to compute (if p and q are known). Then if M' is an encrypted message, the original message M may be recovered by using the fact that

$$M = (M')^d \bmod N ,$$

(provided the process started with $M < N$), so the private key process is identical to the public key process except that e has been replaced by d.

An apparent difficulty in the method is the calculation of $M^e \bmod N$ and $(M')^d \bmod N$. The message M is broken into pieces, each of which is smaller than N, but this number will have well over one hundred digits in an actual application. The process of repeated multiplication will require some computational effort. The number e is often chosen to be 65537, which is relatively small, but d could be any number in the approximate range $\frac{N}{e}$ to N, so a very long multiplication process is involved. Fortunately, exponentiation can be done with relative ease by repeated squaring, and the number of multiplication operations will never be more than $O(\log_2 N)$ (a few hundred, in most applications). However, the private key process is still many times as much work as the public key process, although hardware has been developed which enables very fast decryption rates to be attained.

The function f_{RSA} is a trapdoor function because there is no known way of duplicating the private key process without knowledge of d, and there is no known way of discovering d from e and N, except by factoring N. Surely the only way of finding $(M')^d \bmod N$, without knowing d, is to try all possibilities, until a sensible message is obtained. As d will have well over 100 digits, factorising N is a simpler task. Several hundred years of searching for rapid methods of factorisation, by renowned mathematicians, have not revealed any easy way to do this task. Furthermore, the difficulty of factorisation increases rapidly as □ increases in size, so that future increases in computational power can be matched by choosing N to be a little larger. It has not been proven mathematically that the discovery of d from e and N, is equivalent to factorising N (except in special cases), but it is commonly accepted that, even if it is not equivalent to factorisation, any method of enabling d to be computed easily would be of tremendous mathematical significance, probably with far-reaching ramifications in number theory, and is highly unlikely.

2.1 The mathematics of the RSA System

The Euler totient function of N, denoted $\varphi(N)$, is the number of numbers less than N which are relatively prime to N (ie have no common factor with N). If $N = pq$ where p and q are different primes, then

$$\varphi(N) = (p-1)(q-1).$$

If e is any number relatively prime to $\varphi(N)$, then by the extended Euclidean algorithm, a number d can be found such that

$$ed = k\varphi(N) + 1, \text{ for some integer } k; \qquad (2.1)$$

this is simply another way of writing the identity given above, $ed \equiv 1 \bmod \varphi(N)$.

When the public key process is applied to a number M, followed by the private key process, we obtain

$$(M^e \bmod N)^d \bmod N = M^{ed} \bmod N.$$

Similarly, if the private key process is applied to M, followed by the public key process, we obtain the same expression, $M^{ed} \bmod N$. Because of equation (2.1), M^{ed} is equal to

$$M^{ed} = M^{k\varphi(N)+1} = (M^{\varphi(N)})^k \times M. \qquad (2.2)$$

The RSA method works because, if M is relatively prime to N, then $M^{\varphi(N)} \bmod N$ is 1, by a well-known theorem of Euler. (The probability of M not being relatively prime to N is around 10^{-50}, for any likely N - it effectively "can't happen", in that exposure of the private key due to unforseen circumstances is much more likely.) Hence, if M is less than N,

$$\begin{aligned} M^{ed} \bmod N &= (M^{\varphi(N)})^k \times M \bmod N && \text{by (2.2)} \\ &\equiv 1^k \times M \bmod N \\ &= M\,. && (2.3) \end{aligned}$$

Note that the number $\varphi(N)$ can be replaced by the least common multiple of $p-1$ and $q-1$; the proof above needs a trivial change in one place. In practice, however, $p-1$ and $q-1$ will not have a large common factor, for otherwise N would be easy to factorise. The proof can be easily modified to cope with the case where N is the product of more than two prime numbers (as long as they are all different), but this does not increase the security of the encryption/ decryption process. A final point of interest is the symmetry between d and e; if p and q are known, then either one can be found from the other. If it was desired to have a quicker private key process, a short d could be chosen (not too short, though), and the corresponding e worked out and published.

2.2 A weakness of the RSA System

One weakness of the RSA system is that signature forging is possible if particular messages, chosen beforehand by the intending forger, are signed by the victim (see Goldwasser, Micali and Rivest [2]). This is called an "adaptive chosen-message attack" and it is possible because of the multiplicative nature of the trapdoor function used. Since

$$M^d L^d = (ML)^d ,$$

if M, $M^d \bmod N$, L and $L^d \bmod N$ are known, then both ML and $(ML)^d \bmod N$ can be computed without knowledge of d. Note that $M^d \bmod N$ is the signature attached to M by the private key process. If M and L were very carefully chosen so that ML was the message which the forger wished to sign, then the signature of ML can be generated by multiplying the signatures of M and L. This scenario is admittedly rather unlikely, but in order to prevent it a hash function is normally applied to M before the signature is computed; the hash function of ML is not the product of the hash functions of M and L, so this forging method breaks down.

It should be noted, however, that the use of a hash function does not seem to prevent forging. It is possible to proceed as follows:

- Choose a message, hash it if a hash function is being used, and factorise the result into primes.

- Generate lots of innocent messages, hash them if necessary, factorise the results, and choose those which contain one of the primes obtained in the first step (or in a previous version of this step).
- Obtain the signatures of the messages chosen in the second step.
- Use an elimination process to obtain the signatures of the constituent primes.
- Multiply these signatures together to get the signature desired.

A very large number of innocent messages would be needed in practice; it appears to be simpler just to factorise N. However, there are other ways of obtaining the signatures of a set of primes, and this attack could become feasible at some point. The point we wish to make is that the use of a hashing technique is not a real barrier to forging; the RSA system has a fundamental weakness because of its multiplicative nature, which cannot be resolved by hashing. No such weakness is known for LUC.

3. The LUC Public Key System

We now explain our new public key system. It is based on a different trapdoor function from the RSA and El Gamal systems, which is defined by Lucas functions (described in section 3.1). Because the properties of Lucas functions mirror those of exponentiation, public key and private key processes can be developed in an exactly analogous manner to the RSA system. This enables us to prove that any successful attack on the LUC system would give a successful attack on the RSA system. Since the weakness of the RSA system described in section 2.2 does not occur for the LUC system, the LUC system is cryptographically stronger than RSA.

3.1 Lucas Functions

Lucas functions are an example of higher order linear recurrences. If $P_1, P_2, P_3, \ldots, P_m$ are integers, then we can define a sequence of integers $\{T_n\}$ by:

$$T_n = P_1 T_{n-1} + P_2 T_{n-2} + \ldots + P_m T_{n-m} \,.$$

We have to define $T_0, T_1, T_2, \ldots, T_{m-1}$ independently, in order to be able to use the defining equation. This equation is called an m'th order linear recurrence relation. It is easy to see that a sequence defined by a first-order linear recurrence relation consists of numbers which are a constant (T_0) times successive powers of P_1. Sequences satisfying higher order linear relations can be thought of as generalisations of powers, so it is not too surprising that a generalisation of the RSA system to some of these sequences is possible. At present, we only have a generalisation to sequences satisfying second-order linear recurrence relations, and we will restrict the discussion to these relations from here on.

The general second-order linear recurrence relation, in the form in which we will consider it, is :

$$T_n = PT_{n-1} - QT_{n-2} \,. \qquad (3.1)$$

We will always take P and Q to be relatively prime integers. If we take $P = 1 = -Q$, then the sequence of numbers obtained by choosing $T_0 = 0$ and $T_1 = 1$ is the well-known Fibonacci sequence.

The general form of a sequence obtained from a second-order linear recurrence relation can be found easily. Let α and β be the roots of the polynomial equation

$$x^2 - Px + Q = 0 \,. \qquad (3.2)$$

If c_1 and c_2 are any numbers, then the sequence $\{c_1\alpha^n + c_2\beta^n\}$ has the property that

$$
\begin{aligned}
P(c_1\alpha^{n-1} + c_2\beta^{n-1}) - Q(c_1\alpha^{n-2} + c_2\beta^{n-2}) &= c_1\alpha^{n-2}(P\alpha - Q) + c_2\beta^{n-2}(P\beta - Q) \\
&= c_1\alpha^{n-2}(\alpha^2) + c_2\beta^{n-2}(\beta^2)
\end{aligned}
$$

by (3.2)

$$= c_1\alpha^n + c_2\beta^n .$$

So this sequence satisfies the second-order linear recurrence relation (3.1), and it is not difficult to see that any sequence $\{T_n\}$ satisfying (3.1) must be of the form $\{c_1\alpha^n + c_2\beta^n\}$, where $T_0 = c_1 + c_2$, $T_1 = c_1\alpha + c_2\beta$

Note that if T_0 and T_1 are integers, then by (3.1), all the terms in the sequence will be integers, even though α, β, c_1 and c_2 are (probably) not integers, and may not even be real.

There are two particular solutions of the general second-order linear recurrence relation which are of particular interest. They are denoted by $\{U_n\}$ and $\{V_n\}$, and are defined by:

$$U_n = \frac{\alpha^n - \beta^n}{\alpha - \beta} \qquad (\text{so } c_1 = \frac{1}{\alpha - \beta} = -c_2)$$

$$V_n = \alpha^n + \beta^n \qquad (\text{so } c_1 = 1 = c_2).$$

These will both be sequences of integers, since we have:

$$U_0 = 0,\ U_1 = 1,\ V_0 = 2,\ V_1 = P .$$

These sequences depend only on the integers P and Q, and the terms are called the Lucas functions of P and Q. They are sometimes written $U_n(P,Q)$ and $V_n(P,Q)$, to emphasise their dependence on P and Q. They were first discussed by Lucas in [5]. A paper by Lehmer [4] extended the theory of these functions considerably. A more recent reference is the book by Ribenboim [8].

Note that if N is any number, then

$$U_n(P \bmod N, Q \bmod N) \equiv U_n(P,Q) \bmod N ,$$

because this result is certainly true when n is 0 or 1, and for every n which is 2 or greater,

$$U_n(P,Q) \bmod N \equiv (P \bmod N)(U_{n-1}(P,Q) \bmod N) - (Q \bmod N)(U_{n-2}(P,Q) \bmod N),$$

so the stated result follows by induction. Similarly

$$V_n(P \bmod N, Q \bmod N) \equiv V_n(P,Q) \bmod N . \qquad (3.3)$$

3.2 Lucas Function Relationships

Since the roots of (3.2), α and β, satisfy the equations

$$\alpha + \beta = P, \quad \alpha\beta = Q,$$

it is not difficult to find many relationships between the Lucas functions U_n and V_n, and the coefficients of the recurrence relation (3.1), P and Q. The discriminant of (3.2), $D = P^2 - 4Q$, can be expressed in terms of α and β by:

$$D = (\alpha - \beta)^2.$$

We will need the following relationships, which are easy to obtain by using the definitions of U_n, V_n, D and Q in terms of α and β :

$$V_{2n} = V_n^2 - 2Q^n \tag{3.4}$$

$$V_{2n-1} = V_n V_{n-1} - PQ^{n-1} \tag{3.5}$$

$$V_{2n+1} = PV_n^2 - QV_n V_{n-1} - PQ^n \tag{3.6}$$

$$V_n^2 = DU_n^2 + 4Q^n \tag{3.7}$$

$$2V_{n+m} = V_n V_m + DU_n U_m \tag{3.8}$$

$$2Q^m V_{n-m} = V_n V_m - DU_n U_m \tag{3.9}$$

Consider the linear recurrence relation created by using $V_k(P,Q)$ for P and Q^k for Q:

$$T_n = V_k(P,Q)T_{n-1} - Q^k T_{n-2}.$$

The roots of the corresponding quadratic equation, α' and β', must satisfy

$$\alpha' + \beta' = V_k(P,Q) = \alpha^k + \beta^k \quad \text{and} \quad \alpha'\beta' = Q^k = \alpha^k\beta^k,$$

so we must have $\alpha' = \alpha^k$ and $\beta' = \beta^k$ (or vice versa). This means that

$$V_n(V_k(P,Q),Q^k) = (\alpha^k)^n + (\beta^k)^n = \alpha^{nk} + \beta^{nk} = V_{nk}(P,Q)$$

This composition result is crucial; it is a clear generalisation of the rule for composition of powers, with the subscript of a Lucas function playing the role of a power. If we take $Q = 1$, then we get the simple relationship

$$V_{nk}(P,1) = V_n(V_k(P,1),1) \,. \tag{3.10}$$

Lucas functions have been used to prove divisibility results (see Williams [10]), because there are relationships between the subscript of a Lucas function and the divisors of its value. To explain these, we need to define the Legendre symbol:

$\left(\frac{D}{p}\right)$ = 0 if p divides D, otherwise

= 1 if there is a number x such that $D \equiv x^2 \bmod p$, or

= −1 if no such number exists.

If p is an odd prime number which does not divide Q or D, and ε is $\left(\frac{D}{p}\right)$, then Lehmer proved in [4] (see also Williams [10]) that

$$U_{k(p-\varepsilon)}(P,Q) \equiv 0 \bmod p \text{ for any integer } k \,. \tag{3.11}$$

It is also true that

$$V_{k(p-\varepsilon)}(P,Q) \equiv 2Q^{k(1-\varepsilon)/2} \bmod p \text{ for any integer } k \,; \tag{3.12}$$

this can be derived from Lehmer's article, in the same way as (3.11), for k = 1, and the result for general k follows by induction from

$$V_{(k+1)(p-\varepsilon)} = \frac{1}{2}(V_{k(p-\varepsilon)}V_{p-e} + DU_{k(p-\varepsilon)}U_{p-\varepsilon}) \qquad \text{by (3.8)}$$

$$\equiv V_{k(p-\varepsilon)}Q^{(1-\varepsilon)/2} \bmod p \qquad \text{by (3.11)}.$$

With any sequence of Lucas functions defined by mutually prime integers P and Q, there is a generalisation of the Euler totient function for Lucas functions, the Lehmer totient function (see [4]). Its general definition, and a full explanation of its relationship to the Euler totient function, are not our concern here. We will only need to apply it to numbers N of the form $N = pq$, with p and q different odd primes. In this case, the Lehmer totient function of N is

$$T(N) = (p - \left(\frac{D}{p}\right))(q - \left(\frac{D}{q}\right)) \,.$$

Just as in the case of the Euler totient function, the full product is not needed for the results we will use, and we only need the least common multiple of the factors; writing lcm for least common multiple, we define

$S(N) = \mathrm{lcm}((p - \left(\frac{D}{p}\right)),(q - \left(\frac{D}{q}\right)))$.

Since $S(N)$ is a product of both $(p - \left(\frac{D}{p}\right))$ and $(q - \left(\frac{D}{q}\right))$, the results (3.11) and (3.12) show that, when $N = pq$, p and q different odd primes not dividing D ($= P^2 - 4$ in this case):

$U_{kS(N)}(P,1) \equiv 0 \bmod N$ for any integer k (3.13)

and $V_{kS(N)}(P,1) \equiv 2 \bmod N$ for any integer k. (3.14)

If $N = pq$ is a product of two different odd primes, $P < N$ is relatively prime to N, $P^2 - 4$ is also relatively prime to N, e is any number relatively prime to $S(N)$, and d is found (by the extended Euclidean algorithm) so that:

$ed = kS(N) + 1$, for some integer k, then we obtain

$$
\begin{aligned}
V_d(V_e(P,1),1) &= V_{de}(P,1) && \text{by (3.10)}\\
&= V_{kS(N)+1}(P,1)\\
&= PV_{kS(N)}(P,1) - V_{kS(N)-1}(P,1) && \text{by (3.1)}\\
&= PV_{kS(N)}(P,1) - (1/2)(V_{kS(N)}(P,1)V_1(P,1) - DU_{kS(N)}(P,1)U_1(P,1)) && \text{by (3.9)}\\
&\equiv (2P - (1/2)(2P - 0)) \bmod N && \text{by (3.13) and (3.14)}\\
&= P. && (3.15)
\end{aligned}
$$

This result, a clear analogy to (2.3), enables us to construct a bijective trapdoor function using Lucas functions.

One apparent difference with the situation in the case of powers is an apparent lack of symmetry between the function $V_e(P,1)$ and its inverse $V_d(R,1)$. The numbers d and e are related through the function $S(N)$, which involves quadratic residues with respect to D, which is defined in terms of P. By symmetry, it would be expected that $S(N)$ should have the same value, regardless of whether P or $V_e(P,1)$ is used in its definition. By (3.7) we have:

$V_e^2(P,1) - 4 = DU_e^2(P,1)$.

Also, $\left(\frac{D}{p}\right) = \left(\frac{DU_e^2}{p}\right)$, since Legendre symbols are not changed by squares, as can be easily seen from the definition above. This means

$$\left(\frac{D}{p}\right)=\left(\frac{P^2-4}{p}\right)=\left(\frac{V_e^2(P,1)-4}{p}\right),$$

so the value of $S(N)$ is the same, regardless of whether it is computed for the direct function or the inverse function.

3.3 The New Public Key System, LUC

Using the results in the previous paragraph, a public key system can be developed by analogy with the RSA system. Suppose N and e are two chosen numbers, with N the product of two different odd primes, p and q. The number e must be chosen so it is relatively prime to $(p-1)(q-1)(p+1)(q+1)$. Let M be a message which is less than N and relatively prime to N (as pointed out in section 2.1, this is not a real restriction on M). We define

$$f_{LUC}(M)=V_e(M,1) \bmod N$$

where V_e is a Lucas function, as defined in section 3.1. This is the LUC public key process, giving an encrypted message, M'. To define the matching private key process, we need a number d such that

$$de \equiv 1 \bmod S(N)$$

where $S(N)=\operatorname{lcm}\left(\left(p-\left(\frac{D}{p}\right)\right),\left(q-\left(\frac{D}{q}\right)\right)\right)$,

where $D=(M')^2-4$, and $\left(\frac{D}{p}\right)$, $\left(\frac{D}{q}\right)$ are the Legendre symbols of D with respect to p and q. We can assume that D is relatively prime to N (again, not a real restriction), so the Legendre symbols are either +1 or −1. Hence the original choice of e ensures that it is relatively prime to $S(N)$, so the number d can be found easily by the extended Euclidean algorithm. The private key process is then the same as the public key process, with e replaced by d. By (3.15) and (3.3), and the fact that $M<N$,

$$M=V_d(V_e(M,1) \bmod N,1) \bmod N,$$

and the private key process and public key process are inversions of each other by the symmetry between e and d.

There are two ways in which this process appears to have difficulties; firstly, the computation of V_e and V_d looks extremely long, for large values of e or d, and secondly, the private key number d needs to be recomputed for each message. Both of these difficulties are not serious, as we now explain.

Computation of Lucas functions can be done by a successive doubling technique, as in the "Russian peasant" method of multiplication (see Knuth [3, 4.6.3]). This technique is due to Williams ([10]). All arithmetic is done modulo N, and Q is 1 for the Lucas functions being used, so that equation (3.4) can be written

$$V_{2n}(M,1) \bmod N = ((V_n(M,1) \bmod N)^2 - 2) \bmod N ,$$

and similarly equations (3.5) and (3.6) have modified forms.

If e has the binary expansion

$$e = \sum_{i=0}^{t} x_i 2^{t-i} \qquad (x_0 = 1, x_i = 0 \text{ or } 1 \text{ if } i > 0),$$

then let e_k be the partial sum $\sum_{i=0}^{k} x_i 2^{t-i}$, so that e_t is e and e_0 is 1. Define

$$R_k = V_{2e_k - 1}(M,1) \bmod N, \quad S_k = V_{2e_k}(M,1) \bmod N, \quad T_k = V_{2e_k + 1}(M,1) \bmod N .$$

Then for each k, R_k, S_k and T_k can be computed from $V_{e_k}(M,1) \bmod N$ and $V_{e_k - 1}(M,1) \bmod N$, using the modified forms of (3.4), (3.5) and (3.6). The values of $V_{e_{k+1}}(M,1) \bmod N$ and $V_{e_{k+1} - 1}(M,1) \bmod N$ can then be obtained from R_k, S_k and T_k using the fact that $e_{k+1} = 2e_k$, $e_{k+1} - 1 = 2e_k - 1$ if $x_{k+1} = 0$, while $e_{k+1} = 2e_k + 1$, $e_{k+1} - 1 = 2e_k$ if $x_{k+1} = 1$.

This method ensures that V_e and V_d can be computed in about the same length of time as the e and d powers are computed in the RSA method. Having to compute two numbers at each stage does slow the computation down a little, but there are optimisations in the calculation which mean that the total amount of computation is only about 50% more than the amount needed for the RSA system.

Because the correct private key needs to be obtained for each decryption, there is a little more work involved in the LUC private key system than in the RSA private key system. Firstly note that the "recomputation" of the private key number d is more apparent than real, as there are only four possible values for d. This follows from the fact that there are four possible values for $S(N)$:

$\operatorname{lcm}((p+1),(q+1))$,

$\operatorname{lcm}((p+1),(q-1))$,

$\operatorname{lcm}((p-1),(q+1))$,

$\operatorname{lcm}((p-1),(q-1))$.

These values are all known at the time that N is first created, so four different values of d can be calculated at that time. Given a message to which to apply the private key process, it is necessary to find the quadratic residues $\left(\frac{(M')^2 - 4}{p}\right)$ and $\left(\frac{(M')^2 - 4}{q}\right)$. This can be done by an algorithm analogous to the Euclidean algorithm, which will take around $O(\log_2 p) + O(\log_2 q)$ operations, about the same order as the number needed for the calculation of $V_d(M',1) \bmod N$. The p and q calculations are much easier, however, because p and q have about half as many digits as N, and in practice the quadratic residue calculations take up to 20% of the time needed for the Lucas function calculation. Once the quadratic residues are known, the correct d value can be used in the Lucas function calculation, giving a total amount of computational work about 80% more than that needed for the RSA private key process.

One way of avoiding the need to compute the quadratic residues is to define a new function

$$R(N) = \operatorname{lcm}((p-1),(q-1),(p+1),(q+1)),$$

and choose d so that

$$de \equiv 1 \bmod R(N).$$

Since $\boxed{\quad}$ is a multiple of all possible $S(N)$ values, the calculations leading to (3.15) are unchanged, and the d obtained in this way would work in all cases. The only problem is that it would probably have around twice as many digits as N, so the computational effort to calculate $V_d(M',1) \bmod N$ for this d will be almost exactly doubled.

If sufficient computing power is available, the quadratic residue calculations may be done in "no" extra time. The computation of $V_d(M',1) \bmod N$, for all four possible values of d, can proceed independently on four processors at the same time as the quadratic residues are being computed on another processor. The correct private key processed form can be selected once the correct quadratic residues are known, and the other three computations discarded.

The RSA exponentiation calculations may be speeded up by various heuristic methods. These methods turn out to be applicable to Lucas function calculations, with a few changes. It appears that the calculation time for LUC will always be roughly comparable to the calculation time for RSA.

3.3 Cryptographic Strength of LUC

Just as for the RSA method, the LUC private key process can be discovered only if there is a way of computing $V_d(M',1) \bmod N$ without knowledge of d, or if there is a way of finding d from e and N. The second problem is harder than the corresponding problem for the RSA method, because there are four different values of d for each pair of e and N, only one of which will work for an arbitrary M'. The first problem is really no different from the problem of computing powers; trial of all possible values seems to be the only way.

The fact that Lucas functions are a generalisation of powers makes it certain that any successful attack on the LUC public key system would automatically lead to a successful attack on the RSA public key system. Because of the additional complications with Lucas functions, however, the reverse may not be true; successful attack on RSA may not lead to a successful attack on LUC. For example, the weakness of RSA, due to its multiplicative nature, is not shared by LUC. Thus we say, with confidence, that LUC is cryptographically stronger than RSA.

The discussion in the previous section shows that the computational effort required for the LUC public key process is about the same as that required for the RSA public key process, while the LUC private key process involves less than double the computational effort of the RSA private key process (and may take considerably less, if parallel computation is available). Some signature formation time could be saved by omitting the hashing process, since LUC is not susceptible to adaptive chosen-message attacks, but in practice a simple hashing method would be used to compress the message before signing.

3.4 A new version of the Diffie-Hellman Method

The first public key application described was the Diffie-Hellman key negotiation method ([1]). This method enables two correspondents to agree upon the same private number, using only public information. Each publishes a number of the form $\alpha^X \bmod q$, where q is a fixed prime number, α is a fixed primitive root, and X is private. Since

$$\left(\alpha^X \bmod q\right)^Y \bmod q = \left(\alpha^Y \bmod q\right)^X \bmod q = K$$

both can obtain the same number K without revealing their private numbers X and Y. The security of the method is based on the discrete logarithm problem; if q and X are large then computation of X from $\alpha^X \bmod q$ is infeasible.

This method has an alternative in Lucas functions. A large prime q is agreed upon as above, and also a number α with the property that

$$\left(\frac{\alpha}{q}\right) = -1 \text{ and } k/(q+1),\ V_k(\alpha,1) \equiv 2 \bmod q \ \Rightarrow\ k = q+1 .$$

Then numbers of the form $V_X(\alpha,1) \bmod q$ are published. Since

$$V_Y(V_X(\alpha,1) \bmod q, 1) \bmod q = V_X(V_Y(\alpha,1) \bmod q, 1) \bmod q ,$$

as proved above, this method works in the same way as the Diffie-Hellman method. Little research appears to have been published on the Lucas function equivalent of the discrete logarithm problem, so the security of this method is not known, but it appears likely that it is at least as strong as the Diffie-Hellman method.

Acknowledgments

We would like to thank Chris Skinner and Gary Tee for assistance in the early development of this paper, Marston Conder and Horst Gerlach for providing an alternative proof of the LUC method, and for numerous helpful conversations, and Hugh Williams and Rudolph Lidl for pointing out earlier work in this field.

References

[1] W. Diffie, M. Hellman, "New directions in cryptography", *IEEE Transactions on Information Theory*, **22** (1976) pp644-654.

[2] S. Goldwasser, S. Micali, R. L. Rivest, "A digital signature scheme secure against adaptive chosen-message attacks", *SIAM Journal of Computing*, **17** (1988) pp281-307.

[3] D. E. Knuth, "Seminumerical Algorithms", Addison-Wesley (1981).

[4]D. H. Lehmer, "An extended theory of Lucas' functions", *Annals of Math.*, **31** (1930) pp419-448.

[5] F. E. A. Lucas, "Théorie des fonctions numériques simplement périodiques", *American Jnl Math.*, **1** (1878) pp184-240, 289-321.

[6] W. B. Müller, W. Nöbauer, "Some remarks on public-key cryptosystems", *Studia Sci. Math. Hungar.,* **16** (1981) pp71-76.

[7] W. B. Müller, W. Nöbauer, "Cryptanalysis of the Dickson-scheme", *Advances in Cryptology: Proceedings of Eurocrypt '85,* Springer-Verlag Lecture Notes in Computer Science **219** (1986) pp50-61.

[8] P. Ribenboim, "The book of prime number records", Springer-Verlag (1988).

[9] R. L. Rivest, A. Shamir, L. Adelman, "A method for obtaining digital signatures and public-key cryptosystems", *Comm. ACM*, **21** (1978) pp120-126.

[10] H. C. Williams, "A p+1 method of factoring", *Mathematics of Computation*, **39** (1982) pp225-234.

[11] H. C. Williams, "Some public-key crypto-functions as intractable as factorization", *Cryptologia*, **9** (1985) pp223-237.