Cryptography using Chebyshev polynomials

G. J. Fee and M. B. Monagan
Centre for Experimental and Constructive Mathematics,
Simon Fraser University,
Burnaby, Canada, V5A 1S6
gfee@cecm.sfu.ca and mmonagan@cecm.sfu.ca

Abstract

We consider replacing the monomial x^n with the Chebyshev polynomial $T_n(x)$ in the Diffie-Hellman and RSA cryptography algorithms. We show that we can generalize the binary powering algorithm to compute Chebyshev polynomials, and that the inverse problem of computing the degree n, the discrete log problem for $T_n(x) \bmod p$, is as difficult as that for $x^n \bmod p$.

1 Introduction

If Alice wants to send a secret message to Bob, using a conventional secret key cryptographic algorithm such as the DES (Data Encryption Standard) [1] [2] [4], or the AES (Advanced Encryption Standard) [3], then Alice and Bob must first agree on a common secret key. If Alice emails Bob the secret key, a wiretapper might copy the key and decrypt the secret message that Alice sent to Bob. How can Alice and Bob agree on their common secret key? The Diffie-Hellman [1] [2] [4] key agreement protocol solves this problem.

The Diffie-Hellman Key Agreement Algorithm

1. Alice creates positive integers g and prime p such that $g < p$.
2. Alice chooses a secret integer exponent m such that $0 < m < p$.
3. Alice calculates $a = g^m \bmod p$.
4. Alice emails p, g, and a to Bob.
5. Bob chooses n, a secret integer exponent such that $0 < n < p$.

6. Bob calculates $b = g^n \bmod p$.
7. Bob emails b to Alice.
8. Alice computes the secret key $k = c$ with $c = b^m \bmod p$.
9. Bob computes the secret key $k = d$ with $d = a^n \bmod p$.

The common secret key k is c for Alice and d for Bob, but $c = d$ as $(g^n)^m = g^{nm} = (g^m)^n \bmod p$. In the above protocol, one uses binary powering modulo p to compute quickly the powers g^m, g^n, b^m, and a^n modulo p. As the base and exponent are bounded by p, one can compute the powers in $O(\log p)$ multiplications and divisions, with numbers that contain at most $O(\log p)$ bits in their binary representation. Using classical algorithms we can multiply or divide in $O(\log(p)^2)$ bit operations, this yields a time complexity of $O(\log(p)^3)$ bit operations to compute $g^m \bmod p$, which corresponds to polynomial time in the length of the input.

We can assume that Eve, an eavesdropper knows p, g, a, and b, and needs to compute the secret key k which is either c or d in the above protocol. One method for doing this is to solve the discrete log problem $a = g^m \bmod p$ for secret exponent m, and then to compute $c = b^m \bmod p$, just as Alice would do in step 8. As known methods to solve the discrete log problem have a much larger time complexity, it is presently computationally infeasible to solve the discrete log problem for the secret exponent. For example, for both the Pollard ρ algorithm and the index calculus method [4], the time complexity is $O(\sqrt{p})$, which is of exponential time in $\log(p)$ the length of the inputs. For this reason the above algorithm hinders eavesdroppers from discovering the secret key.

For maximum security it is recommended in [4] that p should be a *safeprime*, that is $(p-1)/2$ should also be a prime, and that g should be a primitive root of p so that the period of the sequence of powers of g is maximal. In current practice p is chosen to have at least 200 decimal digits.

The generalized Diffie-Hellman Key Agreement Algorithm

This algorithm has been generalized from the multiplicative group of integers modulo p, a prime, to any finite group, such as the multiplicative group of non-zero elements in the finite field $GF(2^k)$ [3], the group of points on an elliptic curve under addition [1] [4] [3] or the XTR public key system [5]. Alice chooses a group G and a random element g from the group. As the

binary powering algorithm can be used in any group, both Alice and Bob can quickly compute a common secret group element. For this procedure to be useful, we also require that known algorithms for the discrete log problem be of exponential time in the length of the inputs. One simple example for which this condition does not hold is in the additive group of integers modulo p, for which binary powering becomes multiplication modulo p, and the discrete log problem is solved by doing one division modulo p.

In this work we replace the polynomial x^n that appears in the Diffie-Hellman key agreement protocol with the Chebyshev polynomial of the first kind $T_n(x)$ and show also how to use this in the RSA algorithm. In section 3 we show how to compute $T_n(g) \bmod p$ rapidly. This is the analogue of computing $g^n \bmod p$ using a binary decomposition of n. In section 4 we study properties of the sequence of values of $T_n(x) \bmod p$, and in section 5 we show that the discrete log problem for Chebyshev polynomials can be mapped into the classical discrete log problem. Finally, in section 6, we give an RSA encryption algorithm based on Chebyshev polynomials.

2 Diffie-Hellman Key Agreement with Chebyshev polynomials

We generalize the Diffie-Hellman key agreement protocol as follows. Instead of generalizing the basic rule of exponents $(g^m)^n = g^{mn} = (g^n)^m$ to an arbitrary group, we consider it as a polynomial identity $(x^m)^n = x^{mn} = (x^n)^m$, in which x is an indeterminate. We ask the question: Can polynomials in any class other than the pure monomial x^n can satisfy such a commutative composition identity? A similar identity holds for Chebyshev polynomials of the first kind:

$$T_m(T_n(x)) = T_{mn}(x) = T_n(T_m(x)).$$

We see that this holds by using the classical formula [6], $T_n(x) = \cos(n \arccos(x))$ which is valid for all real x in the interval $[-1, 1]$. Using this formula we have

$$\begin{aligned} T_m(T_n(x)) &= \cos(m \arccos(\cos(n \arccos x))) \\ &= \cos(mn \arccos x) \\ &= \cos(nm \arccos x) \\ &= \cos(n \arccos(\cos(m \arccos x))) \\ &= T_n(T_m(x)) \end{aligned}$$

For $x > 1$ we can use the formula $T_n(x) = \cosh(n \cosh^{-1} x)$ to verify the identity. Up to a linear transformation the pure monomial x^n and the Chebyshev polynomials are the only classes of polynomials that satisfy the commutative composition relation $P_n(P_m(x)) = P_m(P_n(x))$ with $P_n(x)$ a polynomial of degree n. See pages 33-34 exercise E.5 in reference [7] for a proof. Thus we present the Diffie-Hellman Key Agreement algorithm using Chebyshev polynomials:

1. Alice creates positive integers g and prime p such that $g < p$.
2. Alice chooses a secret integer degree m such that $0 < m < p$.
3. Alice calculates $a = T_m(g) \bmod p$.
4. Alice emails p,g, and a to Bob.
5. Bob chooses a secret integer degree n such that $0 < n < p$.
6. Bob calculates $b = T_n(g) \bmod p$.
7. Bob emails b to Alice.
8. Alice computes the secret key $k = c$ with $c = T_m(b) \bmod p$.
9. Bob computes the secret key $k = d$ with $d = T_n(a) \bmod p$.

The integer c for Alice and the integer d for Bob constitute the common secret key k as both have computed $T_{mn}(g) \bmod p$.

Example 1: Here is a simple example of this algorithm in which $m = 2$ and $n = 3$ are chosen hence we need to evaluate $T_2(x) = 2x^2 - 1$, $T_3(x) = 4x^3 - 3x$, and $T_6(x) = 32x^6 - 48x^4 + 18x^2 - 1$.

1. Alice chooses $p = 89$ and $g = 7$.
2. Alice chooses $m = 2$.
3. Alice computes $a = T_2(7) = 2(7^2) - 1 \bmod 89 = 97 \bmod 89 = 8$.
4. Alice emails Bob $p = 89$, $g = 7$, and $a = 8$.
5. Bob chooses $n = 3$.
6. Bob computes $b = T_3(7) = 4(7^3) - 3(7) \bmod 89 = 1351 \bmod 89 = 16$.
7. Bob emails $b = 16$ to Alice.
8. Alice computes $c = T_2(16) = 2(16^2) - 1 \bmod 89 = 511 \bmod 89 = 66$.
9. Bob computes $d = T_3(8) = 4(8^3) - 3(8) \bmod 89 = 2024 \bmod 89 = 66$.

Both Alice and Bob have generated the same secret key $k = 66$, which is also $T_6(7) \bmod 89 = 3650401 \bmod 89 = 66$. To make the above protocol practical we require an efficient algorithm for computing $T_n(x) \bmod p$, with both n and p large. We discuss methods for doing this in the next section. For maximum security p should be a safeprime and g should be chosen such that the period is large. We study this in the following section. For p a prime

greater than three, five known cases in which the period is small occur when g is one of $0, 1, (p-1)/2, (p+1)/2$ and $p-1$, with periods of length $4, 1, 3, 6$ and 2 respectively.

3 Binary powering for Chebyshev polynomials

We seek to generalize the binary powering algorithm, so that we can quickly compute values of the Chebyshev polynomials modulo p. First we rewrite the recurrence for Chebyshev polynomials $T_{n+1}(x) = 2xT_n(x) - T_{n-1}(x)$ from [8] formula 22.7.4, page 782, as a matrix equation:

$$\begin{bmatrix} T_n(x) \\ T_{n+1}(x) \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -1 & 2\,x \end{bmatrix} \begin{bmatrix} T_{n-1}(x) \\ T_n(x) \end{bmatrix}$$

This equation with n diminished by 1 becomes

$$\begin{bmatrix} T_{n-1}(x) \\ T_n(x) \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -1 & 2\,x \end{bmatrix} \begin{bmatrix} T_{n-2}(x) \\ T_{n-1}(x) \end{bmatrix}$$

Combining the above equations yields

$$\begin{bmatrix} T_n(x) \\ T_{n+1}(x) \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -1 & 2\,x \end{bmatrix}^2 \begin{bmatrix} T_{n-2}(x) \\ T_{n-1}(x) \end{bmatrix}$$

We continue to replace the vector on the right side until the index is 0, yielding

$$\begin{bmatrix} T_n(x) \\ T_{n+1}(x) \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -1 & 2\,x \end{bmatrix}^n \begin{bmatrix} T_0(x) \\ T_1(x) \end{bmatrix}$$

As $T_0(x) = 1$ and $T_1(x) = x$, the vector on the right is just the transpose of the row vector $[1, x]$. Hence we just compute the above using matrix binary powering, perform one matrix-vector product, and then extract the first element of the resulting vector to obtain $T_n(x)$. Using the classical matrix-matrix multiplication algorithm, we can do one matrix multiplication with individual elements reduced modulo p using 8 integer multiplications, 4 integer additions, and 4 integer remainder operations. As the exponent $n < p$

we use $O(\log(p))$ matrix multiplications modulo p to compute the value of the degree n Chebyshev polynomial modulo p. This calculation would take about 4 to 8 times more C.P.U. time than the conventional Diffie-Hellman algorithm, depending on the cost of a large integer division relative to a large integer multiplication.

We can improve the above binary matrix powering algorithm by computing λ^n modulo the characteristic polynomial of the above recurrence relation matrix. As the determinant is 1 and as the trace is $2x$, the characteristic polynomial is $\lambda^2 - 2x\lambda + 1$. The Cayley-Hamilton theorem states that a matrix satisfies its own characteristic polynomial, so instead of powering a matrix, we just compute the indeterminate λ^n modulo the characteristic polynomial using binary powering. A basic step in the above algorithm would be multiplying two linear polynomials, finding the remainder from the division by the characteristic polynomial, and then reducing the coefficients modulo p. We can multiply two linear polynomials using four integer multiplications of single-length numbers and one integer addition of double-length numbers, in which a single-length number is one that is less than p and a double-length number is one that is less than p^2. The remainder is calculated by doing a substitution for λ^2, which would cost another multiplication and 2 additions. This other multiplication would be a product of a double-length number and a single-length number, so we might reduce the double-length number by performing a division by p before doing the other multiplication. Whether we do this reduction modulo p would depend on the relative cost of our division algorithm and our multiplication algorithm. In practice we implement both versions, time them, and select the faster version. Then we would reduce the two coefficients modulo p at a cost of two integer divisions. If we omitted the intermediate reduction, one of our divisions would have a triple-length number divided by a single-length number. We have found the above algorithm to be slightly quicker than the other matrix powering algorithm.

Timings

We implemented the classical binary powering algorithm, computing $g^n \bmod p$, the Chebyshev matrix powering algorithm, computing $T_n(g) \bmod p$, and the Chebyshev characteristic polynomial algorithm, computing $T_n(g) \bmod p$, with $g = 7^{233}$, $n = 13^{178}$ and $p = 97^{100}+528$. The calculations were performed with Maple 8 [9], on a 180 MHz Silicon Graphics computer.

algorithm	time in seconds
binary powering	0.686
matrix powering	4.527
characteristic polynomial	3.197

The Chebyshev matrix powering algorithm requires 6.60 times the duration of the classical binary powering algorithm, and the other Chebyshev algorithm 4.66 times that duration.

4 Properties of Chebyshev polynomial sequences modulo a prime

To investigate the difficulty of inverting $x = T_n(g) \bmod p$, with n unknown, we generate some experimental data. Choosing a small p, we can compute the sequence $T_n(x) \bmod p$ for $n = 0, 1, 2, ...$ until we discover the period of the sequence. The result is that the period is at most $p + 1$ for any given input argument $x = 0, 1, 2, ..., p - 1$. For example, when $x = 3$ and $p = 11$, the sequence $T_n(3) \bmod 11$ for $n = 0, 1, 2, ..., 23$ is:

$$1, 3, 6, 0, 5, 8, 10, 8, 5, 0, 6, 3, 1, 3, 6, 0, 5, 8, 10, 8, 5, 0, 6, 3$$

which is of period 12. By doing this for each $x = 0, 1, 2, ..., 10$ we find the following sequences:

x	sequence
0	1, 0, 10, 0, 1, 0, 10, 0, 1, 0, 10, 0
1	1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1
2	1, 2, 7, 4, 9, 10, 9, 4, 7, 2, 1, 2
3	1, 3, 6, 0, 5, 8, 10, 8, 5, 0, 6, 3
4	1, 4, 9, 2, 7, 10, 7, 2, 9, 4, 1, 4
5	1, 5, 5, 1, 5, 5, 1, 5, 5, 1, 5, 5
6	1, 6, 5, 10, 5, 6, 1, 6, 5, 10, 5, 6
7	1, 7, 9, 9, 7, 1, 7, 9, 9, 7, 1, 7
8	1, 8, 6, 0, 5, 3, 10, 3, 5, 0, 6, 8
9	1, 9, 7, 7, 9, 1, 9, 7, 7, 9, 1, 9
10	1, 10, 1, 10, 1, 10, 1, 10, 1, 10, 1, 10

Table 1: Example 2

with periods $4, 1, 10, 12, 10, 3, 6, 5, 12, 5, 2$ respectively.

Theorem 1 Let p be an odd prime and $g \in \mathbb{Z}$ such that $0 \leq g < p$. Let k be the period of the sequence $T_n(g) \bmod p$ for $n = 0, 1, 2, \ldots$. Let $\lambda^2 - 2g\lambda + 1$ have roots $\lambda = \alpha_1, \alpha_2$.
Then (i) $k|p-1$ if the roots are in $GF(p)$, otherwise, (ii) $k|p+1$ when the roots are in $GF(p^2)$.

Proof: We have already shown that we can compute $T_n(g) \bmod p$ by computing the n^{th} power of the recurrence relation matrix which has characteristic polynomial $f(\lambda) = \lambda^2 - 2g\lambda + 1$, performing one matrix-vector product modulo p, and then selecting the first element in the resulting vector. The determinant of this matrix is also equal to the coefficient of λ^0 of $f(\lambda)$, which is 1; thus the matrix is non-singular. As the 0^{th} power of the matrix is the identity matrix, the period k is the smallest positive integer such that the k^{th} power of the matrix is the identity matrix. We can compute powers of a matrix by finding its Jordan canonical form, computing powers of the Jordan canonical form, then transforming back. Recall that the Jordan canonical form is either a diagonal matrix with distinct eigenvalues on the diagonal, or an upper triangular matrix with repeated eigenvalues on the diagonal; these eigenvalues are also roots of the characteristic polynomial. In the case that the characteristic polynomial has repeated roots, they both must be $\pm 1 \bmod p$ because the only least-residue solutions of $x^2 - 1 = 0 \bmod p$ are $x = \pm 1$. In the case that 1 is a double root, $g = 1$ and our sequence is $1, 1, 1, ...$ which has period 1. In the other case, the repeated root is -1 with $g = -1$; our sequence is $1, p-1, 1, p-1, ...$ which has period 2 and is a divisor of $p-1$, as p is odd. We consider the case that the distinct roots lie in $GF(p)$. From Fermat's theorem, $a^{(p-1)} = 1 \bmod p$ for any non-zero $a < p$. The $(p-1)^{st}$ power of the Jordan canonical form matrix is the identity matrix, so our sequence has period at most $p - 1$. If a smaller exponent $k < p - 1$ exists such that the k^{th} power of our Jordan canonical form matrix is the identity, k must be a divisor of $p - 1$, because, by Lagrange's theorem for the order of a subgroup of a finite group, the period must be a divisor of $p - 1$. The other case is that the roots exist only in a quadratic extension of $GF(p)$. We can still diagonalize our matrix as long as we do our arithmetic in that quadratic extension field. First we raise our diagonal Jordan canonical form matrix to power p; such raising a number to the power p is an automorphism in the quadratic extension field. Hence the two roots just swap position on the diagonal, as we have only one non-identity automorphism in a quadratic extension field. Then we multiply by our original Jordan canonical-form ma-

trix to complete the calculation of the $(p+1)^{st}$ power. This just multiples two pairs of conjugate roots. As the product of the conjugate roots is also the coefficient of λ^0 in the characteristic polynomial, which is 1, the $(p+1)^{st}$ power is the identity matrix. Thus the period is at most $p+1$. Again, by Lagrange's theorem, the period must be a divisor of $p+1$. ■

We recommend choosing p to be a safeprime, and check that $p+1$ has a large prime factor or that it is hard to factor.

We also noticed a mirror symmetry about the midpoint in each sequence in example 2, which occurs because the roots are reciprocals of each other. Because of this mirror symmetry only about half the elements can occur in any given sequence. As our sequences contain some duplicates before the period is complete, our sequence cannot be explained as being obtained from the powers of an element from a group. For this reason we have a generalization of the Diffie-Hellman algorithm different from the group generalization, even though a matrix group occurs in our algorithm.

5 The discrete log problem for Chebyshev polynomials

How does an eavesdropper try to break our algorithm? We can assume that the eavesdropper knows p, g, a, b but not the secret degrees m and n. One way that an eavesdropper can break our algorithm would be first to solve the equation $a = T_m(g) \bmod p$ for the unknown degree m. We call this the Chebyshev discrete log problem. Then the eavesdropper would just compute $c = T_m(b) \bmod p$ the same way that Alice would. We map the Chebyshev discrete log problem onto the conventional discrete log problem by recalling the hyperbolic definition of Chebyshev polynomials and solving it for m. We have the following theorem.

Theorem 2 Let a and g be integers and p a prime. If $a = T_m(g) \bmod p$ then m is one of the values $\log_{g+\sqrt{g^2-1}}(a+\sqrt{a^2-1})$ in which the square roots lie in the quadratic extension field $GF(p^2)$ and the logarithm is the discrete log in the field $GF(p)$ or its quadratic extension field.

Proof: $a = T_m(g) = \cosh(m \cosh^{-1}(g))$ so $m = \frac{\cosh^{-1}(a)}{\cosh^{-1}(g)}$. We convert this to logarithms to find $m = \frac{\log(a+\sqrt{a^2-1})}{\log(g+\sqrt{g^2-1})}$. Recalling the change of base formula, we

write this as $m = \log_{g+\sqrt{g^2-1}}(a + \sqrt{a^2-1})$. In the case that both the square roots $\sqrt{g^2-1}$ and $\sqrt{a^2-1}$ exist in $GF(p)$ we have a conventional discrete log problem; otherwise, at least one square root exists in the quadratic extension field $GF(p^2)$, which yields a quadratic extension field generalization of the discrete log problem. To compute a square root in $GF(p)$ or in $GF(p^2)$ we can use the $O(\log^3 p)$ probabilistic polynomial time method of Rabin, described in [4].

Example 3 Alice chooses $p = 9973$, $g = 65$ and $m = 1871$. Alice computes $a = T_m(g) \bmod p = 1063$. The eavesdropper has seen p, g, a but not m and needs to determine m. The eavesdropper computes

$$h = \sqrt{g^2-1} = \pm 923, \text{ and } s = \sqrt{a^2-1} = \pm 3296.$$

The eavesdropper then computes the discrete logarithms

$$\log_{g \pm h}(a \pm s) = \pm 1871 \bmod 9972.$$

The eavesdropper has also seen $b = T_n(g) \bmod p$ that was sent from Bob to Alice. He proceeds to compute $k = T_m(b) \bmod p$ and obtains the secret key. Both values of m yield the same key k.

Example 4 Alice chooses $p = 9973$, $g = 73$ and $m = 1259$. Alice computes $a = T_m(g) \bmod p = 1659$. The eavesdropper has seen p, g, a but not m and needs to determine m. The eavesdropper attempts to compute $h = \sqrt{g^2-1} \bmod p$, which exists only in the quadratic extension field $Z_p(\sqrt{5328})$. Next the eavesdropper attempts to compute $s = \sqrt{a^2-1} \bmod p$, but this may be expressed as $s = 5200\sqrt{5328}$. The eavesdropper then computes the discrete logarithms

$$\log_{g \pm h}(a \pm s) = \pm 1259 \bmod 9974.$$

6 The Chebyshev polynomial RSA algorithm

Recall the RSA cryptographic algorithm [4].

1. Alice chooses two large primes p and q which are kept secret.
2. Alice computes the public modulus $M = pq$.
3. Alice chooses a random encryption exponent $0 < e < M$.
4. Alice computes the secret modulus $L = (p-1)(q-1)$.

5. Alice computes a secret decryption exponent $d = 1/e \bmod L$.
6. Alice emails Bob M and e.
7. Bob encodes his secret message as a number x, such that $0 \leq x < M$.
8. Bob computes $y = x^e \bmod M$ and sends y to Alice.
9. Alice decrypts Bob's y by computing $z = y^d \bmod M$.

Now Alice has $z = x$ and then decodes z to read Bob's message.

We modify the above algorithm to use Chebyshev polynomials instead of monomials, as follows.

1. Alice chooses two large primes p and q which are kept secret.
2. Alice computes the public modulus $M = pq$.
3. Alice chooses a random encryption degree $0 < e < M$.
4. Alice computes the secret modulus $L = (p^2 - 1)(q^2 - 1)$.
5. Alice computes a secret decryption degree $d = 1/e \bmod L$.
6. Alice emails Bob M and e.
7. Bob encodes his secret message as a number x, such that $0 \leq x < M$.
8. Bob computes $y = T_e(x) \bmod M$ and sends y to Alice.
9. Alice decrypts Bob's y by computing $z = T_d(y) \bmod M$.

Alice has $z = x$ and then decodes z to read Bob's message. This algorithm works because the period in the sequence of Chebyshev polynomials was shown to be a divisor of $p - 1$ or a divisor $p + 1$, so is always a divisor of the product $p^2 - 1$ for an odd prime modulus p. As our modulus is $M = pq$ we use the Chinese remainder theorem to show that if a message is recovered modulo each of the primes p and q then it can be recovered modulo M. First working modulo p, we find that $z = T_d(y) = T_d(T_e(x)) = T_{de}(x) = T_{k(p^2-1)(q^2-1)+1}(x) = T_1(x) = x \bmod p$. As the same equation holds modulo q Alice can recover Bob's encoded message x. In our case the decryption degree d is about twice the length of the decryption exponent in the classical RSA algorithm. As this condition doubles the decryption time as compared to RSA, the total time should be about 8 to 16 times longer than the time for the conventional RSA algorithm. Our timings in Maple 8 show that encrypting is 4.01 times slower than RSA, and decrypting is 8.39 times slower than RSA, when p is a random 100 digit prime, q is a random 101 digit prime, and e a random 100 digit integer.

7 Appendix

Here are two maple procedures for computing $T_n(x) \bmod m$.

```
Tnm2 := proc(n,x,m)
 local e,a11,a12,a21,a22,s11,s12,s21,s22,r,t1,t2;
 description `Tnm2(n,x,m) computes T[n](x) mod m`,
 `where T[n](x) is the n'th degree Chebyshev`,
 ` polynomial of the first kind`,
 `It use the right-to-left matrix binary powering algorithm`;
 if n=0 then 1;
 elif n=1 then x mod m;
 else
  e := n-1;
  a11 := 1;  a12 := 0;
  a21 := 0;  a22 := 1;
  s11 := 0;  s12 := 1;
  s21 := -1; s22 := 2*x mod m;
  while e>1 do
   e := iquo(e,2,'r');
   if r=1 then
    t1 := a11*s11+a12*s21 mod m;
    a12 := a11*s12+a12*s22 mod m;
    a11 := t1;
    t2 := a21*s11+a22*s21 mod m;
    a22 := a21*s12+a22*s22 mod m;
    a21 := t2;
   fi;
   t1 := s11+s22;
   t2 := s12*s21;
   s11 := s11^2+t2 mod m;  s12 :=   s12*t1 mod m;
   s21 :=   s21*t1 mod m;  s22 := s22^2+t2 mod m;
  od;
  t1 := a21*s11+a22*s21 mod m;
  t2 := a21*s12+a22*s22 mod m;
  t1+t2*x mod m;
 fi;
end:
```

```
Tnm4 := proc(n,x,m)
 local e,a0,a1,s0,s1,x2,r,t1,t2;
 description ‘Tnm4(n,x,m) computes T[n](x) mod m‘,
 ‘where T[n](x) is the n’th degree Chebyshev‘,
 ‘ polynomial of the first kind‘,
 ‘It computes L^n mod the characteristic polynomial‘,
 ‘by using a right-to-left binary powering algorithm‘;
 if n=0 then 1;
 elif n=1 then x mod m;
 else
  e := n-1;
  a0 := 1;  a1 := 0;
  s0 := 0;  s1 := 1;
  x2 := 2*x mod m;
  while e>1 do
   e := iquo(e,2,’r’);
   if r=1 then
    t1 := a1*s1;
    t2 := a0*s0-t1;
    a1 := a0*s1+a1*s0+x2*t1 mod m;
    a0 := t2 mod m;
   fi;
   t1 := s1^2;
   t2 := s0^2-t1;
   s1 := 2*s0*s1+x2*t1 mod m;
   s0 := t2 mod m;
  od;
   t1 := a1*s1;
   t2 := a0*s0-t1;
   a1 := a0*s1+a1*s0+x2*t1 mod m;
   a0 := t2 mod m;
   x*(x2*a1+a0)-a1 mod m;
 fi;
end:
```

References

[1] Alfred J. Menezes, Paul C. van Oorschot and Scott A. Vanstone *Handbook of Applied Cryptography* CRC Press, New York, 1997

[2] Douglas R. Stinson *Cryptography Theory and Practice* CRC Press, New York, 1995.

[3] Douglas R. Stinson *Cryptography Theory and Practice, second edition* Chapman and Hall/CRC Press, Boca Raton, Florida, 2002.

[4] Randall K. Nichols, *ICSA guide to Cryptography*, McGraw-Hill, New York, 1999.

[5] Arjen K. Lenstra and Eric R. Verheul *The XTR Public Key System, in CRYPTO 2000 Proceedings, pages 1-19* Springer, Berlin, 2000.

[6] E.J. Borowski, and J.M. Borwein, *The Harper Collins Dictionary of Mathematics*, Harper Collins Publishers, New York, 1991.

[7] Peter Borwein and Tamas Erdelyi, *Polynomials and Polynomial Inequalities*, Springer, New York, 1995.

[8] M. Abramowitz and A. Stegun, *Handbook of Mathematical Functions*, Dover Publications, New York, 1965.

[9] M.B. Monagan, K.O. Geddes, K.M. Heal, G. Labahn, S.M. Vorkoetter, J. MacCarron *Maple 6 Programming Guide* Waterloo Maple, Waterloo, Canada, 2000.

[10] Arto Salomaa *Public-Key Cryptography, second edition* Springer, Berlin, 1996.