Evaluation Methods for Chebyshev Polynomials

Zhengjun Cao, Lihua Liu, Leming Hong

Abstract. The security of cryptosystems based on Chebyshev recursive relation, $T_n(x) = 2x\,T_{n-1}(x) - T_{n-2}(x)$, relies on the difficulty of finding the large degree of Chebyshev polynomials from given parameters. The relation cannot be used to evaluate $T_n(x)$ if n is very large. We will investigate other three methods: matrix-multiplication-based evaluation, halve-and-square evaluation, and root-extraction-based evaluation. Though they have the same theoretical complexity $O(\log n \log^2 p)$, we find in some cases the root-extraction-based method is more efficient than the others, which is as fast as the general modular exponentiation. The result indicates that the hardness of some cryptosystems based on modular Chebyshev polynomials is almost equivalent to that of solving general discrete logarithm.

Keywords: Chebyshev polynomials; matrix-multiplication-based evaluation; halve-and-square evaluation; root-extraction-based evaluation.

1 Introduction

Chebyshev polynomials are defined by

$$T_n(x) = \cos(n \arccos x),\ \ x \in [-1, 1],\ \ n = 0, 1, 2, \cdots \tag{1}$$

which can make a sequence of orthogonal polynomials, and has a big contribution in the theory of approximation. Chebyshev polynomials have many interesting properties [8, 16, 18, 21]. Since

$$\cos(n \arccos x) + \cos((n-2) \arccos x) = 2\cos(\arccos x)\cos((n-1) \arccos x)$$

we have the general Chebyshev recursive relation,

$$T_0(x) = 1,\ \ \ T_1(x) = x,\ \ \ T_n(x) = 2x \cdot T_{n-1}(x) - T_{n-2}(x),\ \ \ n = 2, 3, \cdots \tag{2}$$

Z. Cao is with Department of Mathematics, Shanghai University, Shanghai, 200444, China. caozhj@shu.edu.cn
L. Liu and L. Hong are with Department of Mathematics, Shanghai Maritime University, Shanghai, 201306, China.

Notice that

$$T_n(T_m(x)) = \cos(n \arccos(\cos(m \arccos x))) = \cos(nm \arccos x) = T_{nm}(x) \tag{3}$$

which is just the so-called semi-group property of Chebyshev polynomials (see Fig.1 for the first 12 Chebyshev polynomials). By the polynomial equality, we have $T_n(T_m(a)) = T_{nm}(a) \bmod N$ for some integers a and N.

```
a = 1; b = x; For[i = 2, i < 13, i++, c = Expand[2 * x * b - a];
 Print["i=", i, "   ", c]; a = b; b = c]
i=2   -1 + 2 x^2
i=3   -3 x + 4 x^3
i=4   1 - 8 x^2 + 8 x^4
i=5   5 x - 20 x^3 + 16 x^5
i=6   -1 + 18 x^2 - 48 x^4 + 32 x^6
i=7   -7 x + 56 x^3 - 112 x^5 + 64 x^7
i=8   1 - 32 x^2 + 160 x^4 - 256 x^6 + 128 x^8
i=9   9 x - 120 x^3 + 432 x^5 - 576 x^7 + 256 x^9
i=10   -1 + 50 x^2 - 400 x^4 + 1120 x^6 - 1280 x^8 + 512 x^10
i=11   -11 x + 220 x^3 - 1232 x^5 + 2816 x^7 - 2816 x^9 + 1024 x^11
i=12   1 - 72 x^2 + 840 x^4 - 3584 x^6 + 6912 x^8 - 6144 x^10 + 2048 x^12
```

Figure 1: The first 12 Chebyshev polynomials

At Eurocrypt'91, Habutsu *et al.* [7] suggested a cryptosystem based on iterating a chaotic map. But Biham [3] pointed out that it was insecure against some attacks. After that, many cryptosystems have been proposed, which were based on the difficulty of finding the large degree of Chebyshev polynomials from given parameters. Kocarev *et al.* [9, 14, 19, 24] have presented some public-key encryptions based on Chebyshev maps. In 2015, Truong *et al.* [23] presented an authentication scheme based on chaotic Chebyshev polynomials in client-server environment. Lawnik and Kapczynski [10] investigated the application of modified Chebyshev polynomials in asymmetric cryptography. In 2019, Li *et al.* [11] proposed an outsourcing scheme for verifiable Chebyshev maps-based chaotic encryptions in the cloud/fog scenarios.

The security of cryptosystems based on Chebyshev polynomials was discussed by Bergamo *et al.* [2, 6, 15]. In 2010, Liao *et al.* [5, 12, 13] suggested some restrictions on the selection of underlying module. Shakiba *et al.* [17, 20, 25] pointed out some flaws of multiplicative coupled cryptosystems based on Chebyshev polynomials. But so far, we have not found any work on evaluating Chebyshev polynomials.

In this paper, we focus on the problem of evaluating modular Chebyshev polynomials, and propose two new algorithms, halve-and-square evaluation, and root-extraction-based evaluation, different from the general matrix-multiplication-based evaluation. We find both halve-and-square method and matrix-multiplication-based method can be efficiently implemented. Though the three methods have the same complexity $O(\log n \log^2 p)$, the root-extraction-based method could be more efficient than the others, if the number x can be expressed as $\frac{a+a^{-1}}{2}$ for some number a. It almost runs as fast as the general modular exponentiation. The result indicates that the security of some systems based on modular Chebyshev polynomials is not always stronger than that of systems based on the general discrete logarithm.

2 Encryptions based on Chebyshev recursive relation

There are many cryptographic applications of Chebyshev polynomials [9, 10, 12, 14, 19, 21–24]. In this section, we only review two kinds of encryptions: one is over the real number field, and the other is over finite fields or rings. The attacks [2, 3, 5, 6, 13, 15, 17, 20, 25] are not true threats to such cryptographic protocols, because the flaws can be successfully eliminated by choosing some proper module to ensure that the period of generated sequence is quite large.

2.1 Encryption over real number field

In 2003, Kocarev and Tasev [9] presented a public key encryption based on Chebyshev recursive relation over real number field, which can be described as follows.

Setup. Pick a large integer s and a random number $x \in (-1, 1)$. Set the public key as $(x, T_s(x))$ and the secret key as s.

Encrypting. Represent a given message as a number $M \in (-1, 1)$. Pick a large integer r to compute the ciphertext $(c_1, c_2) = (T_r(x), M \cdot T_r(T_s(x)))$.

Decrypting. Recover the plaintext $M = c_2/T_s(c_1)$.

Its correctness is due to that $T_s(c_1) = T_s(T_r(x)) = T_r(T_s(x))$. But the equality does not strictly hold because of the involved computational errors.

2.2 Encryption over finite fields or rings

The Kocarev-Tasev encryption can be converted into a counterpart over finite fields or rings. There are many such encryptions [10, 12, 14, 19, 21, 24]. We now only describe the Li *et al.*'s scheme [11].

Setup. Let λ be a secure parameter, q be a λ-bit prime, $H_0 : \mathbb{Z}_q^2 \to \{0,1\}^\lambda, H : \mathbb{Z}_q^2 \times \{0,1\}^\lambda \to \{0,1\}^\lambda$ be two hash functions. Pick $x_0 \in \mathbb{Z}_q$ and $s_0, s_1, s_2 \in \{0,1\}^\lambda$. compute $x_1 = T_{s_0}(x_0), y_1 = T_{s_1}(x_0), y_2 = T_{s_2}(x_1)$, and set the public key as $(q, x_0, x_1, y_1, y_2, H, H_0)$, and the secret key as (s_1, s_2).

Encrypting. For a message $m \in \{0,1\}^\lambda$, pick $r \in \{0,1\}^\lambda$. Compute the ciphertext

$$(c_1, c_2, c_3, c_4) = (T_r(x_0),\ T_r(x_1),\ H_0(T_r(y_1), T_r(y_2)) \oplus m,\ T_r(y_2) \cdot T_\alpha(T_r(y_1))),$$

where $\alpha = H(c_1, c_2, c_3)$.

Decrypting. Compute $\alpha = H(c_1, c_2, c_3)$ and check that $c_4 = T_{s_2}(c_2) \cdot T_\alpha(T_{s_1}(c_1))$. If true, Compute $m = H_0(T_{s_1}(c_1), T_{s_2}(c_2)) \oplus c_3$.

Its correctness is due to that

$$T_{s_1}(c_1) = T_{s_1}(T_r(x_0)) = T_r(T_{s_1}(x_0)) = T_r(y_1),$$
$$T_{s_2}(c_2) = T_{s_2}(T_r(x_1)) = T_r(T_{s_2}(x_1)) = T_r(y_2).$$

3 Evaluating Chebyshev polynomials

The recursive relation Eq.(2) cannot be used to compute $T_n(x)$ if n is very large, because it needs to do $O(n)$ multiplications. So, we need to find other methods for evaluating Chebyshev polynomials. If $x \in (-1, 1)$, the expression $T_n(x) = \cos(n \arccos x)$ can be used to evaluate. The related series representations are

$$\arccos(x) = \pi/2 - x - x^3/6 - (3x^5)/40 - \cdots,$$
$$\cos(x) = 1 - x^2/2 + x^4/24 - \cdots$$

The numerical computational errors for calculating $\arccos x$ must be kept very small. Otherwise, the numerical values between $\cos(n \arccos x)$ and $\cos((n+k) \arccos x)$ for some moderate integer k, cannot be practically distinguished. That is to say, if $n = 2^{160}$, the computational error of calculating $\arccos x$ must be restricted to 2^{-160} at least. In practice, the requirement is too harsh to meet. So, the method is only suitable for moderate n, say, $n \leq 2^{20}$.

3.1 Matrix-multiplication-based evaluation

The Chebyshev recursive relation can be written as

$$\begin{pmatrix} T_n(x) \\ T_{n-1}(x) \end{pmatrix} = \begin{pmatrix} 2x & -1 \\ 1 & 0 \end{pmatrix}^{n-1} \begin{pmatrix} T_1(x) \\ T_0(x) \end{pmatrix} \tag{4}$$

for $n > 1, T_0(x) = 1, T_1(x) = x$.

Theorem 1. *The matrix-multiplication-based evaluation of $T_n(x)$ needs to do $O(\log n)$ number multiplications.*

Proof. Let $b_k b_{k-1} \cdots b_1 b_0$ be the binary string of $n-1$. We have

$$\begin{pmatrix} 2x & -1 \\ 1 & 0 \end{pmatrix}^{n-1} = \left(\cdots \left(\left(\begin{pmatrix} 2x & -1 \\ 1 & 0 \end{pmatrix}^2 \begin{pmatrix} 2x & -1 \\ 1 & 0 \end{pmatrix}^{b_{k-1}} \right)^2 \cdots \begin{pmatrix} 2x & -1 \\ 1 & 0 \end{pmatrix}^{b_1} \right)^2 \begin{pmatrix} 2x & -1 \\ 1 & 0 \end{pmatrix}^{b_0}$$

So, the method using repeated squaring needs to do $\log n$ matrix multiplications, and each requires 8 number multiplications. Thus, it requires $O(\log n)$ number multiplications. If the computations are executed over the finite field $\mathbb{F}_p$, its computational complexity is $O(\log n \log^2 p)$. $\square$

3.2 Halve-and-square evaluation

Notice that

$$\cos(n \arccos x) = \begin{cases} 2 \cdot \cos^2(\frac{n}{2} \arccos x) - 1, & 2 \mid n \\ 2 \cdot \cos(\frac{n+1}{2} \arccos x) \cdot \cos(\frac{n-1}{2} \arccos x) - \cos(\arccos x), & 2 \nmid n. \end{cases}$$

Hence, we have

$$T_n(x) = \begin{cases} 2 \cdot T_{n/2}^2(x) - 1, & 2 \mid n \\ 2 \cdot T_{(n+1)/2}(x) \cdot T_{(n-1)/2}(x) - x, & 2 \nmid n \end{cases} \tag{5}$$

For convenience, we will call it halve-and-square method.

Theorem 2. *The halve-and-square evaluation of $T_n(x)$ needs to do $O(\log n)$ number multiplications.*

Proof. Given the odd degree q, we have

$$\begin{aligned}
& T_q(x) = 2 \cdot T_{(q+1)/2}(x) \cdot T_{(q-1)/2}(x) - x \\
& \quad \text{intermediate values} \;\; T_{(q+1)/2}(x), \;\; T_{(q-1)/2}(x) \\
\xrightarrow{\text{if } 2|(q+1)/2} \; & T_{(q+1)/2}(x) = 2 \cdot T^2_{(q+1)/4}(x) - 1, \; T_{(q-1)/2}(x) = 2 \cdot T_{(q+1)/4}(x) \cdot T_{(q-3)/4}(x) - x \\
& \text{intermediate values} \;\; T_{(q+1)/4}(x), \;\; T_{(q-3)/4}(x) \\
\xrightarrow{\text{if } 2|(q-3)/4} \; & T_{(q+1)/4}(x) = 2 \cdot T_{(q+5)/8}(x) \cdot T_{(q-3)/8}(x) - x, \; T_{(q-3)/4}(x) = 2 \cdot T^2_{(q-3)/8}(x) - 1 \\
& \text{intermediate values} \;\; T_{(q+5)/8}(x), \;\; T_{(q-3)/8}(x) \\
& \cdots
\end{aligned}$$

Clearly, the size of the constructed intermediate structure is a linear function of the recursive depth $\log n$. Thus, the method can be efficiently executed. There are $2 \log n$ intermediate values, and each requires one multiplication. So it needs to do $O(\log n)$ number multiplications. If the method is executed over the finite filed $\mathbb{F}_p$ where p is a prime, then its complexity is $O(\log\, n \log^2 p)$. $\square$

The semi-group property Eq.(3) can also be used to evaluate Chebyshev polynomials. To do so, one needs to factor n. Ultimately, it turns to the evaluation of $T_q(x)$ for some large prime q. In theory, this method can generate a small-size intermediate structure in comparison with the halve-and-square method. But its programming code seems quite difficult to compose.

To implement this method, one needs to find the halve-chain of the degree n. For instance, $n = 1001$, its halve-chain is $\{1, 2, 3, 4, 7, 8, 15, 16, 31, 32, 62, 63, 125, 126, 250, 251, 500, 501, 1001\}$. Then the recursive description can be converted into its iterative version. We have tested the algorithm for the module $p = 2^{521} - 1$, the number $x = 1234567890987654320$, and the degree $n = 1000000000000000000000000000000000001$. The value of $T_n(x) \bmod p$ is

$$39933700749669457314673333144419570099704646367933343453450171058006608418958807849454871069389571730241075085991445492418345301811944504323652576825758932 11.$$

3.3 Root-extraction-based evaluation

If set $x = \frac{a+a^{-1}}{2}$, we have

$$\frac{a^n + a^{-n}}{2} = 2 \times \frac{a + a^{-1}}{2} \times \frac{a^{(n-1)} + a^{-(n-1)}}{2} - \frac{a^{(n-2)} + a^{-(n-2)}}{2} \mod N,$$

$$T_n\left(\frac{a + a^{-1}}{2}\right) = \frac{a^n + a^{-n}}{2} \mod N. \tag{6}$$

So, the intractability of some cryptosystems based on modular Chebyshev polynomials is equivalent to finding n such that $y = \frac{a^n+a^{-n}}{2} \mod N$, for given a, y, N, which is not the standard discrete logarithm.

For the Chebyshev polynomials over $\mathbb{F}_p$ where p is a prime, if x can be expressed as $\frac{a+a^{-1}}{2}$, the evaluation of Eq.(6) could be very fast because it does not need to store any intermediate value, and only involves one modular exponentiation and two inverse elements. Note that

$$x = \frac{(x + \sqrt{x^2 - 1}) + (x + \sqrt{x^2 - 1})^{-1}}{2}.$$

If $x^2 - 1 \bmod p$ has square roots, we have $a = x + \sqrt{x^2 - 1} \bmod p$, and

$$T_n(x) = \frac{(x + \sqrt{x^2 - 1})^n + (x + \sqrt{x^2 - 1})^{-n}}{2} \bmod p \tag{7}$$

The built-in function PowerMod[ ] cannot tackle a large exponent due to overflows. So, we need to design a new function MyPowerMod[ ] by using repeated squaring (See Appendix 1).

There is an efficient algorithm for root extraction, i.e., Adleman-Manders-Miller algorithm [1]. Its basic idea can be described as below. Write $p - 1 = 2^t s, 2 \nmid s$. Given a quadratic residue δ and a quadratic nonresidue ρ, i.e., $(\delta^s)^{2^{t-1}} \equiv 1 \bmod p, (\rho^s)^{2^{t-1}} \equiv -1 \bmod p.$. If $t \geq 2$, then $(\delta^s)^{2^{t-2}} \bmod p \in \{1, -1\}$. Take $k_1 \in \{0, 1\}$ such that $(\delta^s)^{2^{t-2}} (\rho^s)^{2^{t-1} \cdot k_1} \equiv 1 \bmod p$. Since $(\delta^s)^{2^{t-3}} (\rho^s)^{2^{t-2} \cdot k_1} \bmod p \in \{1, -1\}$, take $k_2 \in \{0, 1\}$ such that $(\delta^s)^{2^{t-3}} (\rho^s)^{2^{t-2} \cdot k_1} (\rho^s)^{2^{t-1} \cdot k_2} \equiv 1 \bmod p$. Likewise, take $k_3, \cdots, k_{t-1} \in \{0, 1\}$ such that $\delta^s (\rho^s)^{2 \cdot k_1 + 2^2 \cdot k_2 + \cdots + 2^{t-1} \cdot k_{t-1}} \equiv 1 \bmod p$. Thus,

$$\left(\delta^{\frac{s+1}{2}}\right)^2 \left((\rho^s)^{k_1 + 2 \cdot k_2 + \cdots + 2^{t-2} \cdot k_{t-1}}\right)^2 \equiv \delta \mod p.$$

Its computational complexity is $O(\log^3 p + t^2 \log^2 p)$ (see [4]). Since p is usually set as a strong prime, i.e., $t = 1$, it becomes $O(\log^3 p)$, and $\delta^{\frac{p+1}{2}} \equiv \delta \bmod p$. If $4 \mid p+1$, then $\delta^{\frac{p+1}{4}} \bmod p$ is just a square root of δ modulo p. The evaluation of Eq.(7) needs $O(\log n \log^2 p)$ cost. So, if pick $a \in \mathbb{F}_p$, and compute $x = \frac{a+a^{-1}}{2} \bmod p$, then the method only needs $O(\log n \log^2 p)$ cost.

3.4 Comparisons

Both the matrix-multiplication-based method and halve-and-square method can be used to evaluate Chebyshev polynomials for $x \in (-1, 1)$. For modular Chebyshev polynomials, both methods naturally have to compute the successive values $T_{k-1}(x), T_k(x)$ in each loop, while the root-extraction-based method only needs to compute the value $T_k(x)$ in each loop. See the Table 1 for the comparisons of the three methods.

Table 1: Comparisons of some evaluation methods

	Numerical evaluation	Modular evaluation	Immediate structure	Multiplications in each loop	Complexity
Matrix-multiplication-based method	Yes	Yes	No	8	$O(\log n \log^2 p)$
Halve-and-square method	Yes	Yes	Yes	2	$O(\log n \log^2 p)$
Root-extraction-based method	No	Yes	No	2	$O(\log^3 p)$ or $O(\log n \log^2 p)$

We find there has no any significant performance difference between the methods (Appendix 1). In theory, the root-extraction-based method could be more efficient. If one directly picks a and sets $x = \frac{a+a^{-1}}{2} \bmod p$, this method almost is as fast as the general modular exponentiation. So the claim that the security of systems based on modular Chebyshev polynomials is stronger than that of systems based on the general discrete logarithm is not always sound.

4 Conclusion

In this paper, we propose two new evaluation methods for Chebyshev polynomials. They have the same complexity $O(\log n \log^2 p)$ as the general matrix-multiplication-based method. The explicit expression $T_n(\frac{a+a^{-1}}{2}) = \frac{a^n+a^{-n}}{2}$ indicates that the hardness of some cryptosystems based on modular Chebyshev polynomials is almost equivalent to that of solving general discrete logarithm.

References

[1] L. Adleman, K. Manders, and G. Miller. On taking roots in finite fields. In *Proceedings of 18th Annual Symposium on Foundations of Computer Science*, pages 175–178. IEEE Computer Society, 1977.

[2] P. Bergamo and *et al.* Security of public-key cryptosystems based on chebyshev polynomials. *IEEE Trans. Circuits Syst. I Regul. Pap.*, 52-I(7):1382–1393, 2005.

[3] E. Biham. Cryptanalysis of the chaotic-map cryptosystem suggested at eurocrypt'91. In *Proceeding of Advances in Cryptology - EUROCRYPT '91, Workshop on the Theory and Application of of Cryptographic Techniques, Brighton, UK, April 8-11, 1991*, pages 532–534. Springer, 1991.

[4] Z. Cao, Q. Sha, and X. Fan. Adleman-Manders-Miller root extraction method revisited. In *Proceedings of Information Security and Cryptology - 7th International Conference*, pages 77–85. Springer, 2011.

[5] F. Chen, X. Liao, T. Xiang, and H. Zheng. Security analysis of the public key algorithm based on chebyshev polynomials over the integer ring Z_n. *Inf. Sci.*, 181(22):5110–5118, 2011.

[6] K. Cheong and T. Koshiba. More on security of public-key cryptosystems based on chebyshev polynomials. *IEEE Trans. Circuits Syst. II Express Briefs*, 54-II(9):795–799, 2007.

[7] T. Habutsu and *et al.* A secret key cryptosystem by iterating a chaotic map. In *Proceedings of Advances in Cryptology - EUROCRYPT'91, Workshop on the Theory and Application of of Cryptographic Techniques, Brighton, UK, April 8-11, 1991*, pages 127–140, 1991.

[8] M. Hunziker, A. Machiavelo, and J. Park. Chebyshev polynomials over finite fields and reversibility of automata on square grids. *Theor. Comput. Sci.*, 320(2-3):465–483, 2004.

[9] L. Kocarev and Z. Tasev. Public-key encryption based on chebyshev maps. In *Proceedings of the 2003 International Symposium on Circuits and Systems, ISCAS 2003, Bangkok, Thailand, May 25-28, 2003*, pages 28–31. IEEE, 2003.

[10] M. Lawnik and A. Kapczynski. The application of modified chebyshev polynomials in asymmetric cryptography. *Comput. Sci.*, 20(3), 2019.

[11] J. Li and *et al.* Verifiable chebyshev maps-based chaotic encryption schemes with outsourcing computations in the cloud/fog scenarios. *Concurr. Comput. Pract. Exp.*, 31(22), 2019.

[12] Z. Li, Y. Cui, Y. Jin, and H. Xu. Parameter selection in public key cryptosystem based on chebyshev polynomials over finite field. *J. Commun.*, 6(5):400–408, 2011.

[13] X. Liao, F. Chen, and K. Wong. On the security of public-key algorithms based on chebyshev polynomials over the finite field Z_n. *IEEE Trans. Computers*, 59(10):1392–1401, 2010.

[14] J. Lima, D. Panario, and R. Souza. Public-key encryption based on chebyshev polynomials over GF(q). *Inf. Process. Lett.*, 111(2):51–56, 2010.

[15] J. Lima, R. Souza, and D. Panario. Security of public-key cryptosystems based on chebyshev polynomials over prime finite fields. In *Proceedings of IEEE International Symposium on Information Theory, ISIT 2008, Toronto, ON, Canada, July 6-11, 2008*, pages 1843–1847. IEEE, 2008.

[16] J. Mason. Chebyshev polynomials of the second, third and fourth kinds in approximation, indefinite integration, and integral transforms. *J. Compu. Appl. Math.*, (49):169–178, 1993.

[17] C. Quan and *et al.* Cryptanalysis of a chaotic chebyshev polynomials based remote user authentication scheme. In *Proceedings of International Conference on Information Networking, ICOIN 2018, Chiang Mai, Thailand, January 10-12, 2018*, pages 438–441. IEEE, 2018.

[18] C. Qureshi and D. Panario. The graph structure of chebyshev polynomials over finite fields and applications. *Des. Codes Cryptogr.*, 87(2-3):393–416, 2019.

[19] G. Rajaram, K. Ramar, and P. Pillai. Public key cryptosystems based on chaotic-chebyshev

polynomials. In *Proceeding of International Conference on Advances in Recent Technologies in Communication and Computing, Kottayam, Kerala, India, 27-28 October 2009*, pages 4–8. IEEE Computer Society, 2009.

[20] A. Shakiba, M. Hooshmandasl, and M. Meybodi. Cryptanalysis of multiplicative coupled cryptosystems based on the chebyshev polynomials. *Int. J. Bifurc. Chaos*, 26(7):1–11, 2016.

[21] J. Tirrell and Y. Zhuang. Hopping from chebyshev polynomials to permutation statistics. *Electron. J. Comb.*, 26(3):P3.27, 2019.

[22] T. Truong, M. Tran, and A. Duong. Improved chebyshev polynomials-based authentication scheme in client-server environment. *Secur. Commun. Networks*, 2019:4250743:1–4250743:11, 2019.

[23] T. Truong, M. Tran, A. Duong, and I. Echizen. Chaotic chebyshev polynomials based remote user authentication scheme in client-server environment. In *Proceedings of ICT Systems Security and Privacy Protection - 30th IFIP TC 11 International Conference, SEC 2015, Hamburg, Germany, May 26-28, 2015*, pages 479–494. Springer, 2015.

[24] S. Yan, P. Zhen, and L. Min. Provably secure public key cryptosystem based on chebyshev polynomials. *J. Commun.*, 10(6):380–384, 2015.

[25] D. Yoshioka. Security of public-key cryptosystems based on chebyshev polynomials over Z/pkZ. *IEEE Trans. Circuits Syst. II Express Briefs*, 67-II(10):2204–2208, 2020.

Appendix 1: Wolfram Mathematica codes for three evaluation methods

```
myp = 2^521 - 1; myx = 1 234 567 890 987 654 320;
myn = 100 000 000 000 000 000 000 000 000 000 000 001;
(*---matrix-multiplication-based method---*)
ChebyshevMatrix[mymodular_, myinput_, myiterations_] :=
  Module[{p, x, n, A, B, T, i}, p = mymodular; x = myinput;
   n = myiterations; A = {{1, 0}, {0, 1}}; B = {{2*x, -1}, {1, 0}};
   f[y_] := Mod[y, p]; T = IntegerDigits[n - 1, 2];
   For[i = 1, i < Length[T], i++, If[T[[i]] == 1, A = Dot[A, B]];
    A = Map[f, Dot[A, A], {2}]]; If[Last[T] == 1, A = Dot[A, B]];
   A = Map[f, Dot[A, {x, 1}], {1}]; Print[A[[1]]]];
Timing[ChebyshevMatrix[myp, myx, myn]]
3 993 370 074 966 945 731 467 333 314 441 957 009 970 464 636 793 334 345 345 017 -
 105 800 660 841 895 880 784 945 487 106 938 957 173 024 107 508 599 144 549 241 -
 834 530 181 194 450 432 365 257 682 575 893 211
{0., Null}

myp = 2^521 - 1; myx = 1 234 567 890 987 654 320;
myn = 100 000 000 000 000 000 000 000 000 000 000 001;
(*---halve-and-square method---*)
HalveSquare[mymodular_, myinput_, myiterations_] :=
 Module[{p, x, m, n, t, a, b, T, H, r, i, j, d, s}, p = mymodular;
  x = myinput; m = myiterations; T = {n}; n = 1; t = 0;
  While[Mod[m, 2] == 0, m = m / 2; t = t + 1]; n = m;
  For[i = 1, i < 3000, i++, If[n < 3, Break[]];
   a = (n - 1) / 2; b = (n + 1) / 2;
   If[Mod[b, 2] == 0, T = T ⋃ {b - 1, b}; n = a, T = T ⋃ {a, a + 1}; n = b]];
  H = <|1 → Mod[x, p], 2 → Mod[2*x^2 - 1, p]|>;
  For[i = 3, i ≤ Length[T], i++, If[Mod[T[[i]], 2] == 0, j = T[[i]] / 2;
    r = Mod[2*H[[Key[j]]]*H[[Key[j]]] - 1, p];
    H = Append[H, T[[i]] → r], j = T[[i]] + 1; k = j / 2;
    r = Mod[2*H[[Key[k]]]*H[[Key[k - 1]]] - x, p];
    H = Append[H, T[[i]] → r]]]; d = Last[H];
  If[t == 0, Print[d], For[i = 1, i ≤ t, i++, s = Mod[2*d*d - 1, p];
    d = s]; Print[s]]];
Timing[HalveSquare[myp, myx, myn]]
3 993 370 074 966 945 731 467 333 314 441 957 009 970 464 636 793 334 345 345 017 -
 105 800 660 841 895 880 784 945 487 106 938 957 173 024 107 508 599 144 549 241 -
 834 530 181 194 450 432 365 257 682 575 893 211
{0., Null}

myp = 2^521 - 1; myx = 1 234 567 890 987 654 320;
myn = 100 000 000 000 000 000 000 000 000 000 000 001;
(*---root-extraction-based method---*)
MyPowerMod[mybase_, myindex_, mymodular_] :=
  Module[{b, c, q, T, a, i}, b = mybase; c = myindex;
   q = mymodular; a = 1; T = IntegerDigits[c, 2];
   For[i = 1, i < Length[T], i++, If[T[[i]] == 1, a = a*b];
    a = Mod[a^2, q]]; If[Last[T] == 1, a = a*b];
   a = Mod[a, q]; Return[a]];
ChebyshevPowerMod[mymodular_, myinput_, myiterations_] :=
 Module[{p, x, n, r0, r1, r2, r}, p = mymodular; x = myinput;
  n = myiterations; If[Mod[p, 4] ≠ 3, Abort[]];
  If[MyPowerMod[x^2 - 1, (p - 1) / 2, p] ≠ 1, Abort[]];
  k = (p + 1) / 4; a = Mod[x + MyPowerMod[x^2 - 1, k, p], p];
  r0 = PowerMod[2, -1, p]; r1 = MyPowerMod[a, n, p];
  r2 = PowerMod[r1, -1, p]; r = Mod[r0*(r1 + r2), p];
  Print[r]]; Timing[ChebyshevPowerMod[myp, myx, myn]]
3 993 370 074 966 945 731 467 333 314 441 957 009 970 464 636 793 334 345 345 017 -
 105 800 660 841 895 880 784 945 487 106 938 957 173 024 107 508 599 144 549 241 -
 834 530 181 194 450 432 365 257 682 575 893 211
{0., Null}
```