On a public key encryption algorithm based on Permutation Polynomials and performance analyses

Gurgen Khachatrian Martun Karapetyan

Abstract: *In this paper a modification of public key encryption system presented in [Khachatryan, Kyureghyan, 2015] and performance analysis are presented. As described in [Khachatryan, Kyureghyan, 2015], the permutation polynomial $P(x)$ is declared to be a public polynomial for encryption. A public key encryption of given $m(x)$ is the evaluation of polynomial $P(x)$ at point $m(x)$ where the result of evaluation is calculated via so called White box reduction, which does not reveal the underlying secret polynomial $g(x)$. Our analysis have shown that an attacker may acquire some information about the message, having its cipher-text, in case of using certain values of P(x). So either those values of P(x) must be avoided, or the modification presented in this paper must be used. Our implementation's performance was compared to RSA-2048 implementation of CryptoPP library and it was 3.75x and 133x faster on encryption and decryption operations respectively.*

Keywords: *Permutation polynomials, Public-key encryption, White box reduction..*

ACM Classification Keywords*: E.3 DATA ENCRYPTION - Public key cryptosystems*

Conference topic*: Cryptographic methods and protocols, Reliable and Secure Telecommunications;*

MSC*: 11T71, 94A60*

Introduction

Let $GF(q)$ be the finite field with q elements, where q is a prime or power of a prime. A polynomial $f(x)$ over $GF(q)$ is called a permutation polynomial if an equation $f(x) = r$ for any $r \in GF(q)$ has only one root in $GF(q)$. In [Khachatryan, Kyureghyan, 2015] a new class of permutation polynomials was presented and a public key system with white box implementation was provided. Its security relays on the problem of solving a polynomial equation over $GF(2^n)$ when the field representation polynomial is unknown.

A pioneering work describing DH key exchange by Diffie and Hellman [Diffie, Hellman, 1976] was presented in 1976, which is based on the discrete logarithm problem (DLP). In 1978 another fundamental work by Rivest, Shamir and Adleman [Rivest, Shamir, Adleman, 1978], called RSA cryptosystem was presented, which is based on integer factorization problem. Another important development for public key cryptosystems was the invention of Elliptic curve cryptosystems [Miller, 1986] which are based on the algebraic structure of elliptic curves over finite fields.

In this paper we present a modification of the cryptosystem described in [Khachatryan, Kyureghyan, 2015] and performance analysis.

The paper is organized as follows: In section 2 the public key algorithm and white box implementation described in [Khachatryan, Kyureghyan, 2015] are presented in short. In section 3 modification of the public key system is presented. In section 4 implementation aspects and performance optimizations of the proposed system are discussed. Section 5 concludes the paper.

Public Key encryption algorithm based on permutation polynomials

Let $GF(q)$ be a finite field of characteristic p. For every permutation polynomial $f(x)$ over $GF(q)$, there exists a unique polynomial, $f^{-1}(x)$ over $GF(q)$ such that $f(f^{-1}(x)) = (f^{-1}(f(x)) = x$ called the compositional inverse of $f(x)$.

Let $F(x) = \sum_{u=0}^{n} a_u x^u \in GF(2)$ be a primitive polynomial and $P(x) = \sum_{u=0}^{n} a_u x^{2^u}$ be its linearized 2-associate. Elements of the $GF(2^n)$ will be represented through a primitive polynomial $g(x)$ over $GF(2)$. Then $P(x)$ is a permutation polynomial and an algorithm for finding it's compositional inverse P^{-1} was presented in [Khachatryan, Kyureghyan, 2015].

A primitive polynomial $g(x)$ of degree n over $GF(2)$ is used as the public key encryption private (secret) parameter and a permutation polynomial $P(x)$ as the public parameter.

a) Public key encryption: any message $m(x)$ of the length n as an input (plaintext) and evaluation of the polynomial $P(m(x)) = c(x) \ mod \ g(x)$ as an output (ciphertext). The evaluation operation will be implemented via "White box" evaluation without revealing the polynomial $g(x)$. The output of public key encryption will be $c'(x)$ based on "White box" tables explained later.

b) Private key decryption: Given a ciphertext $c'(x)$ calculate $c(x)$. Compute $P^{-1}(c(x)) = P^{-1}\left(P(m(x)) = m(x) \ mod \ g(x)\right.$. [Khachatryan, Kyureghyan, 2015]

The white box evaluation is used to calculate the value of $c'(x)$ without revealing the value of modulo reduction polynomial $g(x)$ in such a way, that the "owner" of the system can calculate $c(x)$ from it. The evaluation procedure will be as follows: all possible residues $x^N \equiv R_N(x) \ mod \ g(x)$ for $N = 2^i r$, where $i = 1, ..., 128, r = 2k+1, k = 0, 1, ..., 63$ are biased by using random 64 secret polynomials $L_0, L_1, \ldots, L_{63}$ based on another secret polynomial $L(X)$ which are only known to the "owner" of the system. All biased values for residues modulo polynomial $g(x)$ are provided to the public in the following manner:

$$B_N(x) = ((R_N(x) \times L_0(x)) \ mod L(x)) \oplus L_{k+1}(x) \tag{0.1}$$

for any $N = 2^i(2k+1)$. Based on above explanation an encoding procedure will be as follows: The user calculates an evaluation result of the polynomial $P(m(x))$ without any reduction, takes the polynomial $R(x)$ that contains all terms of evaluation for the degrees not exceeding 127, and calculates the modulo two sum of nonzero terms $B_N(x)$ denoted by $\sum B_N(x)$ corresponding to nonzero terms of evaluation result exceeding $N = 127$.

An encrypted message $c'(x)$ then contains two 16 byte vectors including $R(x)$, $\sum B_N(x)$ and another 8 byte vector $B = (b_0, b_1, \ldots, b_{63})$, where $b_k = 0$ if the number of nonzero terms with the same value k in evaluation result is even and $b_k = 1$ otherwise for $N = 2^i(2k+1), k = 0, ..., 63$.

Decoding procedure by the "owner" of the system will be as follows: based on the value $B = (b_0, b_2, \ldots, b_{63})$ and vector $\sum B_N(x)$ the "owner" calculates:

$$R(x) \oplus \left(\sum B_N(x) \oplus \sum_{i=1}^{128} b_i \times L_i(x) \right) \times (L_0(x))^{-1} = c(x) \ mod \ L(x). \text{[Khachatryan, Kyureghyan, 2015]} \tag{0.2}$$

Modification of the Public Key encryption algorithm

A public polynomial $P(x) = \sum_{u=0}^{n} a_u x^{2^u}$ is used in the encryption algorithm described in [Khachatryan, Kyureghyan, 2015]. Our analyses showed, that if $a_u = 0$ for all but one value v in $u = 2..7$ then the attacker may gain some information about the plain-text message $m(x)$ having the value of $R(x)$. For example is $P(x) = x^{2^{127}} + x^2 + x$ is used, then if $m(x) = \sum_{i=0}^{126} a_i x^i$ then $R(x) = \sum_{i=0}^{63} a_i x^{2*i}$, which means that the attacker will easily get the values of a_i for $i = 0..63$. A simple solution to this problem is to use values of $P(x)$ for which $a_u = 0$ for $u = 2..7$.

A modification of the public-key system follows, which will make usage of any value of $P(x)$ secure. We will describe the algorithm for $n = 128$, but it can be easily extended to be used for any value of n.

a) Public key encryption: any message $m(x)$ of the length n as an input (plaintext) and evaluation of the polynomial $P(m(x) * x^{128}) = c(x)\ mod\ g(x)$ as an output (ciphertext). The evaluation operation will be implemented via a "White box" evaluation describe later.

b) Private key decryption: Given a ciphertext $c'(x)$ calculate $c(x)$. Compute

$$P^{-1}(c(x)) * x^{2^{n-8}} = P^{-1}\left(P(m(x) * x^{128}\right) * x^{2^{n-8}} =$$
$$m(x) * x^{128} * x^{2^{n-8}} = m(x) * x^{2^{n-1}} = m(x)\ mod\ g(x).$$

This is true because $x^{2^{n-1}} = 1\ \ mod\ \ g(x)$.

After the modification there are no terms with degrees of $N < 128$ in the evaluation result of the polynomial $P(m(x) * x^{128})$, so there is no need of having an $R(x)$ any more.

Polynomials $L_0, L_1, \ldots, L_{127}$ are generated, and values of $B_N(x)$ are provided publicly for any $N = 2^i(2k+1)$, where $i = 1, ..., 128, r = 2k+1, k = 0, 1, ..., 127$.

An encrypted message $c'(x)$ then contains a 16 byte vector $\sum B_N(x)$ and another 16 byte vector $B = (b_0, b_1, \ldots, b_{127})$, where $b_k = 0$ if the number of nonzero terms with the same value k in evaluation result is even and $b_k = 1$ otherwise for $N = 2^i(2k+1), k = 0, ..., 127$.

Let $P(x) = x^{2^{11}} + x^{2^{35}} + x^{2^{77}}$ and $m(x) = x^3 + x^8$. We have that

$$P(m(x) * x^{128}) = (x^{131} + x^{136})^{2^{11}} + (x^{131} + x^{136})^{2^{35}} + (x^{131} + x^{136})^{2^{77}}$$
$$= x^{131\cdot 2^{11}} + x^{136\cdot 2^{11}} + x^{131\cdot 2^{35}} + x^{136\cdot 2^{35}} + x^{131\cdot 2^{77}} + x^{136\cdot 2^{77}}$$
$$= x^{131\cdot 2^{11}} + x^{17\cdot 2^{14}} + x^{131\cdot 2^{35}} + x^{17\cdot 2^{38}} + x^{131\cdot 2^{77}} + x^{17\cdot 2^{80}}$$

We have that $\sum B_N(x) = B_{131\cdot 2^{11}}(x) \oplus B_{17\cdot 2^{14}}(x) \oplus B_{131\cdot 2^{35}}(x) \oplus B_{17\cdot 2^{38}}(x) \oplus B_{131\cdot 2^{77}}(x) \oplus B_{17\cdot 2^{80}}(x)$ where $B_N(x)$ are defined according to (0.1). Thus we have that for the vector $B = (b_0, b_1, \cdots, b_{63})$ in this case $b_1 = 1$

Decoding procedure by the "owner" of the system will be as follows: based on the value $B = (b_0, b_2, \ldots, b_{127})$ and vector $\sum B_N(x)$ the "owner" calculates:

$$(\sum B_N(x) \oplus \sum_{i=1}^{128} b_i \times L_i(x)) \times (L_0(x))^{-1} = c(x)\ mod\ L(x). \quad (0.3)$$

After calculating $c(x) = P(m(x) * x^{128})$ the âĂIJownerâĂİ of the system can decrypt the message: $P^{-1}(c(x)) * x^{2^{n-8}} = m(x)$, where P^{-1} is the compositional inverse of the polynomial $P(x)$.

Implementation aspects and performance optimizations

Public key encryption of the proposed system requires evaluation of the polynomial $P(m(x) \cdot x^{128})$ and then a modular reduction using white box implementation. The evaluation of $P(m(x) \cdot x^{128})$ will require to count the values of $m(x)^{2^i}$ for all $i = 0..n$, where 2^n is the order of $P(x)$. This will require n squaring of polynomial $m(x)$. Let's denote by t the weight of $P(x)$ and by s the weight of $m(x)$. Then the evaluation of $P(m(x) \cdot x^{128})$ will have $t * s$ terms, so $t * s$ XORs of polynomials will be required to count $\sum B_N(x)$.

Calculation of $c(x)$ from $c'(x)$ will require 128 modulo two additions and one multiplication of polynomials as explained in section 3. Final decryption operation will require to compute $P^{-1}(c(x)) * x^{2^120} = m(x)$, where P^{-1} is a compositional inverse of polynomial $P(x)$. Calculation of $P^{-1}(c(x))$ will require counting the values of $c(x)^{2^i}$ for all $i = 0..127$, and XORing the resulting polynomials. If the weight of $P^{-1}(x)$ is t, then t XORs will be required.

The memory required for storing the white box tables, I.E. the values of $B_N(x)$ will be 16 bytes per value for each $N = 2^i(2k+1)$, where $i = 1, ..., 128, r = 2k+1, k = 0, 1, ..., 127$, resulting to overall $128 \times 128 \times 16$ bytes $= 256$ Kbytes.

Some performance optimization were made to the described algorithm. One can notice, that for $P(x) = \sum_{u=0}^{n} a_u x^{2^u}$ and $m(x) = \sum_{v=0}^{n} a_v x^v$,

$$P(m(x)) = sum_{v=0}^{n} a_v \times P(x^v). \tag{0.4}$$

In a similar way for $m(x) = \sum_{l=0}^{n} a_l x^l$,

$$P^{-1}(c(x)) = sum_{l=0}^{n} a_l \times P^{-1}(x^l). \tag{0.5}$$

So if we precalculate and store the values of $\sum B_N(x)$ for terms in $P(x^v)$ for each $v = 0..127$ and the values of $P^{-1}(x^l)$ for all $l = 0..127$, this will require $128 * 16$ bytes of additional memory for each of encryption and decryption operations, but the performance will be increased dramatically. We also calculate the impact of $P(x^v)$ on array B for each $v = 0..127$, which requires another $128 * 16$ bytes of memory. After calculating and storing these values, instead of doing 128 squarings and $t * s$ XORs for encryption, we'll do no squarings and just s XORs. In decryption we will skip doing the squarings. This optimizations speed up both encryption and decryption operations by about 2.2 times.

Performance tests were ran on Intel Core I5 CPU 1.6 GHZ processor, and the CryptoPP library's RSA-2048 implementation was used for comparison. A single encryption and decryption operations took 0.032ms and 0.041ms respectively for our algorithm, and 0.12ms and 5.46ms respectively for RSA-2048. So the algorithm described was 3.75x faster on encryption and 133x faster on decryption.

Conclusion

In this paper a modification of white box encryption scheme [Khachatryan, Kyureghyan, 2015] based on permutation polynomials has been presented. Implementation aspects and performance optimizations were provided.

Bibliography

[Khachatryan, Kyureghyan, 2015] G. Khachatrian, M. Kuregian, Permutation polynomials and a new public key encryption - accepted for publication in Discrete Applied Mathematics journal- February 2015, 9 pages

[Laigle, Chapuy, 2007] Y. Laigle-Chapuy, Permutation polynomials and applications to coding theory, Finite Fields, Appl.13, 58–70, 2007

[Lidl, Niederreiter, 1983] R. Lidl, Niederreiter, Finite Fields, Addison Wesley, reading, MA, 1983.

[Schwenk, Huber, 1998] J. Schwenk, K. Huber,Public key encryption and digital signatures based on permutation polynomials, Electron, Lett.34 (1998), 759–760.

[Zeirler, 1959] N. Zeirler, Linear recurring sequences, J.Soc.Ind.Appl.Math.7,(1959), 31–48.

[Diffie, Hellman, 1976] W. Diffie and M.E. Hellman, New Directions in Cryptography, IEEE Transactions on Information Theory, Vol. IT-22, Nov.1976, 644–654.

[Rivest, Shamir, Adleman, 1978] R. Rivest, A. Shamir, L. Adleman, A Method for Obtaining Digital Signatures and Public-Key Cryptosystems, Communications of the ACM 21 (2), (1978), 120–126.

[Miller, 1986] V.S.Miller, Use of Elliptic curves in cryptography, Advanced in Cryptology-Crypto-85 Proceedings, Springer-Verlag, (1986), 417–426.

Authors' Information

Gurgen Khachatryan *American University of Armenia*
Yerevan, Armenia
e-mail: gurgenkh@aua.am

Martun Karapetyan *Institute for Informatics and Automation Problems*
National Academy of Sciences of Armenia
Yerevan, Armenia
e-mail: martun.karapetyan@gmail.com