High-speed Implementation of AIM symmetric primitives within AIMer digital signature

Minwoo Lee[1[0000−0002−2356−3055]],
Kyungbae jang[2[0000−0001−5963−7127]],
Hyeokdong Kwon[2[0000−0002−9173−512X]],
Minjoo Sim[2[0000−0001−5242−214X]],
Gyeongju Song[2[0000−0002−4337−1843]], and
Hwajeong Seo[1[0000−0003−0069−9061]]

[1]Department of Convergence Security,
Hansung University, Seoul (02876), South Korea,
[2]Department of Information Computer Engineering,
Hansung University, Seoul (02876), South Korea,
`{minunejip, starj1023, korlethean, minjoos9797, thdrudwn98, hwajeong84}@gmail.com`

Abstract. Recently, as quantum computing technology develops, the importance of quantum resistant cryptography technology is increasing. AIMer is a quantum-resistant cryptographic algorithm that was selected as the first candidate in the electronic signature section of the KpqC Contest, and uses symmetric primitive AIM. In this paper, we propose a high-speed implementation technique of symmetric primitive AIM and evaluate the performance of the implementation. The proposed techniques are two methods, a Mer operation optimization technique and a linear layer operation simplification technique, and as a result of performance measurement, it achieved a performance improvement of up to 97.9% compared to the existing reference code. This paper is the first study to optimize the implementation of AIM.

Keywords: KpqC · AIMer · Cryptography Implementation · post-quantum cryptography.

1 Introduction

As quantum computing technology, known as the next-generation computing environment, develops, it can have high performance that surpasses existing supercomputers and can perform complex operations that were previously impossible in polynomial time, so It is approaching as a great threat to cryptography in the future. PQC (Post-Quantum Cryptography) [1] is a next-generation cryptosystem differentiated from existing cryptosystems that rely on the hardness of integer decomposition problems, and means a quantum-resistant cryptosystem. The system is resistant to Shor Algorithm attacks [2] and is popular worldwide. Accordingly, the National Institute of Standards and Technology (NIST) held a quantum-resistant encryption standardization contest and selected a standard algorithm in 2022 [3]. Following this trend in Korea, the KpqC (Korea Post-Quantum Cryptography) Contest was held. In December 2022, 9 digital signature algorithms and 7 public key algorithms passed Round 1. In this paper, we propose a high-speed implementation of AIMer, one of the KpqC Round 1 candidate digital signature algorithms.

2 Related Works

2.1 AIMer

AIMer digital signature algorithm was developed based on zero-knowledge [4], and is a signature scheme using symmetric primitive AIM and BN++ proof system [5]. In the key generation process, public information (iv, ct) and private key pt that satisfy $ct = AIM(iv, ct)$ are generated through security parameters. In the signature process, the signature σ is output through the private key and public key pair $(pt, (iv, ct))$ and the message m. In the verification process, Accept or Reject is output with the private key and public key pair $(pt, (iv, ct))$, message m and signature σ as input. Basically, AIMer uses a power mapping based S-Box on a binary extension field to improve cryptographic primitives. The AIMer development team focused most on the method of calculating the Groebner basis of the ideal consisting of the most well-known polynomials [6] and XL (eXtended Linearization) [7] defense against the attack of multivariate polynomial systems. AIMer secured compatibility with MPCitH (Multi-Party Computation in the Head) [8], which can calculate results without sharing data between participants by using a one-way function structure. Also, the S-Box used internally is designed based on the Mersenne S-Box. This makes AIMer resistant to algebraic attacks [9].

2.2 AIM

AIM is a symmetric primitive proposed in AIMer. AIM is a one-way function designed to resist algebraic attacks, and has compatibility to support secure multicomputation in hardware. Table 1 shows the parameters used by AIMer. There are three schemes of AIM, AIM-I, AIM-III, and AIM-V, but only AIM-I is dealt with in this paper. AIM is designed with an S-box that calculates powers by Mersenne numbers [10]and a linear layer that performs binary matrix multiplication. Figure 1 shows the encryption process of AIM-I.

3 Implementation Techniques

In this implementation, we propose a technique to reduce the cost by optimizing the operation and simplifying the linear layer operation.

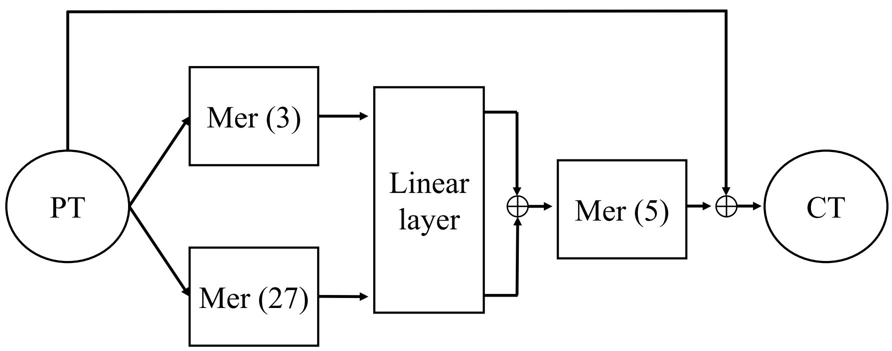

Fig. 1: AIM-I cryptographic process.

Table 1: Parameters of AIM-I.

Scheme	λ	n	l	e_1	e_2	e_3	e_*
AIM-I	128	128	2	3	27	-	5
AIM-III	192	192	2	5	29	-	7
AIM-V	256	256	3	3	53	7	5

3.1 *Mer* operation optimization: *Combined Mer*

The first target of this optimization implementation is an operation performed on the initial, input plaintext. In the case of AIM-1 encryption, the same input value (pt) is copied and operations $Mer(3)$ and $Mer(27)$ are performed, respectively. In this implementation, a single *Combined Mer* operation is proposed instead of performing the operations $Mer(3)$ and $Mer(27)$ respectively. The proposed *Combined Mer* operation has the same complexity as performing the operation of $Mer(27)$ only once. The operation process of the $Mer(3)$ function can be seen in the code in listing 1.

```
void mersenne_exp_3(const GF in, GF out)
{
  GF t1 = {0,};

  // t1 = a ^ (2^2 - 1)
  GF_sqr(in, t1);
  GF_mul(t1, in, t1);

  // out = a ^ (2^3 - 1)
```

```
  GF_sqr(t1, t1);
  GF_mul(t1, in, out);

}

```

Listing 1: $Mer(3)$ operation source code.

In addition, you can check the contents of the *Combined* Mer operation function proposed in the code of listing 2, and you can see that the operation process inside the Mer function is included.

```
void mersenne_exp_27(const GF in, GF out, GF out2)
{
    int i;
      GF t1 = {0,};
      //GF t2 = {0,};
      GF t3 = {0,};

      // t1 = a ^ (2^2 - 1)
      GF_sqr(in, t1);
      GF_mul(t1, in, t1);

      // t2 = a ^ (2^3 - 1)
      GF_sqr(t1, t1);
      GF_mul(t1, in, out2);

      // t3 = a ^ (2^6 - 1)
      GF_sqr(out2, t1);
      GF_sqr(t1, t1);
      GF_sqr(t1, t1);
      GF_mul(t1, out2, t3);

      // t3 = a ^ (2^12 - 1)
      GF_sqr(t3, t1);
      for (i = 1; i < 6; i++)
      {
        GF_sqr(t1, t1);
      }
      GF_mul(t1, t3, t3);

      // t3 = a ^ (2^24 - 1)
      GF_sqr(t3, t1);
      for (i = 1; i < 12; i++)
      {
        GF_sqr(t1, t1);
      }
      GF_mul(t1, t3, t3);

      // out = a ^ (2^27 - 1)
      GF_sqr(t3, t1);
```

```
        GF_sqr(t1, t1);
        GF_sqr(t1, t1);
        GF_mul(t1, out2, out);
}

```

Listing 2: *Combined* Mer operation source code.

$Mer(27)$ The parameters of the function consist of in, out, and $out2$, and represent $state0$ and $state1$ arrays to store the input plaintext and operation results, respectively. The parameters of $Mer(3)$ are in and out, and represent the input plaintext and $state0$, respectively. $Mer(3)$ The result of the operation is stored in the out parameter $state0$, which is also used as a parameter of the $Mer(27)$ operation. Therefore, the same result can be obtained by performing the operation $Mer(3)$ inside the function $Mer(27)$ without performing the operation $Mer(3)$ and then storing the result in $state0$. Operation $Mer(27)$ of the existing reference code proceeded with the operation by storing the operation result of $Mer(3)$ in the $t2$ array for storing intermediate operation values, but the proposed technique stores the result in the $out2$ which is parameter $state0$, not in the $t2$ array.

3.2 Simplify linear layer operations

A look-up table means a set of pre-calculated results for an operation. Using this set, the result value can be obtained faster than the time required for calculation [11]. AIM's linear layer operation performs a total of 4 matrix-vector multiplications and creates a 128*128 binary matrix using the hash value (SHAKE-128) of the initial vector. When this matrix is created, it is not affected by the plaintext, only by the initial vector value iv. Therefore, there is no need to create a matrix each time encryption is performed. Therefore, we propose a method of performing encryption after pre-creating the values of the corresponding 4 matrix using a look-up table. The look-up table is in the form of a two-dimensional array consisting of 128 two arrays each having a size of 64 bits. You can find the proposed lookup table source code in listing 4 of the appendix, which represents the source code for one of the four matrix.

AIM's linear layer consists of two types of linear components: affine layer and feed-forward. The affine layer performs multiplication with derived matrix of matrix A, a random binary matrix of $n * n$ size, and performs addition with $VectorB$, a random constant affected by the initial vector. In this technique, matrix A and vector B are not created because an affine layer is not created. matrix A does not need to be created because it is necessary when constructing the binary matrix specified in the look-up table. Since $VectorB$ performs an addition operation with the $state0$ array before the $Mer(5)$ operation, the corresponding value is required when performing encryption. Accordingly, the value of $VectorB$ also needs to be specified in the form of a constant.

```
    vector_B[1] = 0x9347b8e12b0971a1;
    vector_B[0] = 0xcaf99a30fa2d6733;
```

```
GF_add(state[0], state[1], state[0]);
GF_add(state[0], vector_B, state[0]);
```

Listing 3: Vector B and addition operation source code

In Listing 4, you can check the code that implements the linear layer operation simplification technique by specifying the Vector B value without creating the linear layer. The proposed technique can drastically reduce the encryption operation time by eliminating all costs consumed in the linear layer generation operation.

4 Performance measurement and evaluation

The proposed technique was implemented using the Xcode 14.3 framework, and the reference source code used was the code being distributed by AIMer [4]. However, the reference code have a OpenSSL dependency. So, the code was modified in a stand alone format with the dependency removed. The target processor is an Apple M2 processor running at a maximum speed of 3.49 MHz. After repeating each algorithm 1,000,000 times, the average value of the measured times was used, and the unit is millisecond(ms). The measurement results are shown in Table 2.

Table 2: Performance measurement results.

	Ref.	*Combined Mer*	Linear Layer	*Combined Mer* + Linear Layer
ms	38482	38171	1268	1181
imprv(%)	0	0.91	97.6	97.9

It can be seen that the implementation of the proposed technique shows better performance than the reference code, and in particular, it can be seen that the performance of the implementation to which the linear layer operation simplification is applied shows a big difference. In the case of the implementation using only the Mer operation optimization technique, the performance was improved by 0.81% compared to the reference code. The implementation using only the linear layer operation simplification technique showed 96.7% improved performance, and the implementation using both techniques showed 96.9% improved performance. The degree of performance improvement of the implementation to which the Mer operation optimization technique is applied is measured relatively low, which is considered to be due to the relatively low improvement rate measured because the cost consumed by the linear layer operation is too large. As a result of comparing the implementation with only the linear layer simplification technique and the implementation with both techniques applied, a performance improvement of 7.42% was confirmed. This means that the Mer operation optimization method also produced significant results in cost reduction.

5 Conclusion

In this paper, a high-speed implementation of AIMer, one of the KpqC Round 1 candidate algorithms, was performed. We proposed two techniques, $CombinedMer$, which is a Mer operation optimization technique, and linear layer operation simplification, and showed up to 97.9% better performance than the reference code. In addition, it was confirmed that the Mer operation optimization technique achieved significant performance improvement. Then, in order to further improve the performance of this algorithm, an assembly optimization implementation is performed on ARMv8 to derive additional performance improvement.

References

1. D. J. Bernstein and T. Lange, "Post-quantum cryptography," *Nature*, vol. 549, no. 7671, pp. 188–194, 2017.
2. P. W. Shor, "Algorithms for quantum computation: discrete logarithms and factoring," in *Proceedings 35th annual symposium on foundations of computer science*, pp. 124–134, Ieee, 1994.
3. D. Joseph, R. Misoczki, M. Manzano, J. Tricot, F. D. Pinuaga, O. Lacombe, S. Leichenauer, J. Hidary, P. Venables, and R. Hansen, "Transitioning organizations to post-quantum cryptography," *Nature*, vol. 605, no. 7909, pp. 237–243, 2022.
4. S. Kim, J. Ha, M. Son, B. Lee, D. Moon, J. Lee, S. Lee, J. Kwon, J. Cho, H. Yoon, *et al.*, "Aim: Symmetric primitive for shorter signatures with stronger security," *Cryptology ePrint Archive*, 2022.
5. C. Baum and A. Nof, "Concretely-efficient zero-knowledge arguments for arithmetic circuits and their application to lattice-based cryptography," in *Public-Key Cryptography–PKC 2020: 23rd IACR International Conference on Practice and Theory of Public-Key Cryptography, Edinburgh, UK, May 4–7, 2020, Proceedings, Part I*, pp. 495–526, Springer, 2020.
6. M. Clegg, J. Edmonds, and R. Impagliazzo, "Using the groebner basis algorithm to find proofs of unsatisfiability," in *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pp. 174–183, 1996.
7. D. Lazard, "Résolution des systèmes d'équations algébriques," 1981.
8. Y. Ishai, E. Kushilevitz, R. Ostrovsky, and A. Sahai, "Zero-knowledge from secure multiparty computation," in *Proceedings of the thirty-ninth annual ACM symposium on Theory of computing*, pp. 21–30, 2007.
9. N. T. Courtois and W. Meier, "Algebraic attacks on stream ciphers with linear feedback," in *Advances in Cryptology—EUROCRYPT 2003: International Conference on the Theory and Applications of Cryptographic Techniques, Warsaw, Poland, May 4–8, 2003 Proceedings 22*, pp. 345–359, Springer, 2003.
10. J. A. Solinas *et al.*, *Generalized mersenne numbers.* Citeseer, 1999.
11. K.-w. Wong, "A fast chaotic cryptographic scheme with dynamic look-up table," *Physics Letters A*, vol. 298, no. 4, pp. 238–242, 2002.

6 Appendix

```
uint64_t state0_lower_tr[128][2] = {
        {0x1, 0x0}, {0x2, 0x0}, {0x7, 0x0}, {0x8, 0x0}, {0x1b
    , 0x0}, {0x23, 0x0}, {0x65, 0x0}, {0xbe, 0x0}, {0x1ea, 0
    x0}, {0x363, 0x0}, {0x791, 0x0}, {0xfe9, 0x0}, {0x1495, 0
    x0}, {0x31a9, 0x0}, {0x46df, 0x0}, {0xbc1c, 0x0}, {0
    x183ea, 0x0}, {0x383c3, 0x0}, {0x5c9eb, 0x0}, {0xff44d, 0
    x0}, {0x16dc2e, 0x0}, {0x3d48cd, 0x0}, {0x756621, 0x0},
    {0xeb8633, 0x0}, {0x18357a2, 0x0}, {0x21e4d45, 0x0}, {0
    x6191ca7, 0x0}, {0xda9edfd, 0x0}, {0x10972df1, 0x0}, {0
    x34ab3a98, 0x0}, {0x585678db, 0x0}, {0xee8de9b8, 0x0}, {0
    x1b6ff58c3, 0x0}, {0x2b955f709, 0x0}, {0x7a53d18b1, 0x0},
     {0xd98dc3559, 0x0}, {0x1eee34a12c, 0x0}, {0x3e305db00b,
    0x0}, {0x61d058eeca, 0x0}, {0x9815c483d0, 0x0}, {0
    x1cf953e85f6, 0x0}, {0x328863614ee, 0x0}, {0x6ba4a07cae8,
     0x0}, {0xcdc61bdfc2a, 0x0}, {0x14f6375f3091, 0x0}, {0
    x3e3ea28c7856, 0x0}, {0x5af63fe90759, 0x0}, {0
    xf26d1388fc2e, 0x0}, {0x14611e7d8a792, 0x0}, {0
    x28182058d910c, 0x0}, {0x6eb3c6c2e834c, 0x0}, {0
    x8462eaf1716d3, 0x0}, {0x11b84d2bdd622b, 0x0}, {0
    x36ac12a1056c36, 0x0}, {0x78650fec38cdd3, 0x0}, {0
    xf3e0c7b9a5f8ea, 0x0}, {0x151f47315add797, 0x0}, {0
    x249813356bff641, 0x0}, {0x6f125e4b879344d, 0x0}, {0
    x8d57480cfbeb83d, 0x0}, {0x1fa3270d7545daa9, 0x0}, {0
    x234047f9dc57a00b, 0x0}, {0x682a868ec31aae2a, 0x0}, {0
    xe1d78f09a87b081f, 0x0}, {0xcab4840a68ca8db7, 0x1}, {0
    x4874a0719b867cce, 0x2}, {0x270ac82bf2bd150f, 0x5}, {0
    x5936831dcc91e5fb, 0x8}, {0x18bf66f0fd39c999, 0x1d}, {0
    x730d7506beb1b864, 0x33}, {0xdd1fa26c3607690c, 0x45}, {0
    x8a93c90acc0d7e08, 0xde}, {0xf0e1988bdcdb0271, 0x149}, {0
    x63da59dab61737aa, 0x3d1}, {0x33fb3e194df62abb, 0x666},
    {0x7e66d2e6423945f9, 0xa7b}, {0x39eba14e8aeed580, 0x13f8
    }, {0x48918514babc960a, 0x32a0}, {0xba3c0b700ea8bd15, 0
    x5bf0}, {0x3a1ad16056faab3c, 0xe982}, {0x776c428a8ed28703
    , 0x11d27}, {0x3d5ad6035115667f, 0x36c5c}, {0
    x5c4f698ef84d31bb, 0x6e5e3}, {0x991d199498b1bca1, 0xf7958
    }, {0x9d402dcdb4c4cef8, 0x1e68a7}, {0xfc0f8e7c74205e14, 0
    x2f5a16}, {0xff70ad46a811c206, 0x44d235}, {0
    x8bfc8b1bc0ac4b3e, 0xf9b960}, {0xc60648c57de85836, 0
    x1e62d30}, {0xec1381065d11d213, 0x2f7c29b}, {0
    xb3f1582a95e2a9f6, 0x65eadee}, {0xb8552b058e9c35cf, 0
    x9c0b700}, {0x2d9de24fc597c0bf, 0x1eb63c9f}, {0
    x60153809eed9c43f, 0x32667c4e}, {0x6b0b783bf25750ee, 0
    x77aefee5}, {0x91164c7665027a6c, 0x8f682670}, {0
    x4f142006415b325f, 0x17f31a3de}, {0xf600ae030f478e5, 0
    x2077f518f}, {0x6d439cabc916ddcd, 0x58620b140}, {0
    x4dab212cbc076e9e, 0xb239d78f5}, {0x6d3ed87ff3a75bbd, 0
    x1dbe443692}, {0xff7fd8ba3068400d, 0x2cb6cb82a4}, {0
    x70644b7d2c05333a, 0x5542096a0f}, {0xc2bf86048b522d0b, 0
    xc6fcb97928}, {0x53051f68065d47f, 0x15654445d63}, {0
    x965229d408b067e9, 0x23e9bdf33e0}, {0xde1bfc4588a91825, 0
```

```
x65f33a7a687}, {0xb390a6de9c90544f, 0xa3618b412bd}, {0
x27ff0ea4f71eec82, 0x1c70f6f3ad11}, {0x764634d8af97a7b3,
0x3a1dbf89d2f2}, {0x3a058f18ae17a8f7, 0x4f4df6504a39}, {0
x64bbc52805567b7, 0x82262f8cebe6}, {0xef3acd344735e9b3, 0
x13e7916a367fd}, {0xeee26c6de6756ea8, 0x3d18a958511fc},
{0x790a5b7c9e7777f2, 0x4191e6080d0bd}, {0
x81a08eb8a3d81b0d, 0x8f858208e9630}, {0x845d43afdc0b5409,
 0x10082f05bcf5d9}, {0x42c6ab960cbf9000, 0x2505f037d4521c
}, {0x83a4431e922f898d, 0x43dd4ecbe65126}, {0
xc94636296b9f9d5c, 0xa098c7ae0be674}, {0x128b104e8032c714
, 0x1a24dc3050c3b9a}, {0xf4df6f12494d6329, 0
x2e5b2e9a667f55d}, {0x3baac65368115806, 0x6432206732ea67d
}, {0x4a7d9f4b717853d5, 0x89183b9005a247a}, {0
xe90463a993e75e58, 0x12c70277dc8280c1}, {0
x9cc9d553a6f70252, 0x3a02a242c974cacb}, {0
xc6ab94d98d676c64, 0x5812810cec0bbf3b}, {0
x5dd04bf4dafda543, 0xf992b0c6783c4b09}

 };


```

Listing 4: source code of a lookup table for one of the four matrix.