High Speed High Assurance implementations of Mutivariate Quadratic based Signatures

Extended Abstract

Samyuktha M[1,2], Pallavi Borkar[1], and Chester Rebeiro[1]

[1] Indian Institute of Technology, Madras
[2] Society for Electronic Transactions and Security, Chennai
samyuktha@setsindia.net
{pallavi,chester}@cse.iitm.ac.in

Abstract. In this poster, we present a Jasmin implementation of Mayo2, a multivariate quadratic(MQ) based signature scheme. Mayo overcomes the disadvantage of the Unbalanced oil and vinegar(UOV) scheme by whipping the UOV map to produce public keys of sizes comparable to ML-DSA . Our Jasmin implementation of Mayo2 takes 930 μs for keygen, 3206 μs for sign, 480 μs for verify based on the average of 1,00,000 runs of the implementation on a 2.25GHz x86 64 processor with 256 GB RAM. To this end, we have a multivariate quadratic based signature implementation that is amenable for verification of constant-time, correctness, proof of equivalence properties using Easycrypt. Subsequently, the results of this endeavor can be extended for other MQ based schemes including UOV.

Keywords: Formal Verification · Mayo · Jasmin

1 Introduction

With the widespread migration of security protocols to post-quantum, various efficient and architecture-specific optimized implementations have emerged. These implementations can be considerably complex and their correctness not easily verifiable due to larger input spaces. Thus, these implementations need to be proven correct and equivalent to their corresponding algorithmic specification in order to achieve the high levels of assurance needed for critical embedded system applications like root of trust, remote attestation and secure communication.

In the context of the NIST PQC competition, it is essential to formally investigate the security of the submissions and their implementations against side-channel attacks. These attacks can compromise a mathematically strong algorithm by exploiting information leaked through side channels such as power consumption and electromagnetic radiation of the device executing the cryptographic operation. A potent side-channel variant called the timing attack utilizes execution differences to reveal information about the secret key. Thus, it is crucial not only to verify the correctness of the cryptographic implementation but also ensure constant time execution.

Multivariate cryptographic schemes based on the oil and vinegar (OV) problem have been well studied and are of interest to the research community as they offer shorter signatures and faster verification times. With NIST standardizing ML-DSA, ML-KEM and an additional call for proposals to focus on signatures based on hardness other than lattices, evaluation of these schemes is relevant.

We plan to carry out the following three-fold approach: (i) developing high-speed formally verified MQ based signature scheme implementations, (ii) verification for functional correctness, and (iii) verification for constant-time execution. In line with the strategy, we have implemented the Keygen, Sign and Verify primitives of a MQ based signature scheme (Mayo2 [7]) in Jasmin, the specifics of which will be detailed in the later sections. Hence we make the following contributions in this article:

1. Bitsliced implementation of the underlying finite field arithmetic
2. Bitsliced implementation of the linear algebra operations
3. Bitsliced Echelon form implementation to solve the linear system of equations
4. Jasmin implementation of the Keygen, Sign and Verify primitives of Mayo2

The code is made available at `https://github.com/samyukthasets/Mayo2-Jasmin`

Related Work. Formally verified implementations, often called high-assurance cryptographic software, exist for classical algorithms and have been widely adopted in libraries. It is imperative to extend these notions to post-quantum schemes for arguing robust security guarantees. Frameworks like Easycrypt [6] utilize model checking and have demonstrated effectiveness in the formal verification process. In their research on a formally verified implementation of SHA-3, Almeida et al. [3] advocate for mechanized proofs of functional correctness, provable security, and resistance to timing attacks, employing a toolchain that combines Jasmin [4] and Easycrypt. Furthermore, Almeida et al in [5] present a formally verified proof of the functional correctness and IND-CCA security of ML-KEM, the Kyber-based Key Encapsulation Mechanism (KEM).

2 Preliminaries

Multivariate Quadratic based Schemes. Signature schemes based on the hardness of multivariate quadratic(MQ) problem offer shorter signature sizes and faster verification times. UOV is one such well-studied MQ based scheme but it suffers from the disadvantage of larger public key sizes. Mayo can be considered as a variant of UOV offers public keys whose size are comparable to that of ML-DSA. However, it is to be noted that the cryptographic security of Mayo depends on the security of UOV.

Mayo. The keygen(1^λ) of Mayo starts by sampling a random matrix $\mathbf{O}$ and the column space of $\begin{bmatrix}\mathbf{O}\\ I_0\end{bmatrix}$ forms the secret oil space $\mathcal{O}$ with $\dim(O) = o$; number of variables 'n', number of equations 'm' and $o < m$. A random multivariate quadratic map $\mathcal{P}(x)$: $\mathbb{F}_q^n \rightarrow \mathbb{F}_q^m$ which vanishes on this subspace $\mathcal{O}$ is generated as the public key.

In a smaller subspace where o < m, with larger probability the system will not have any solution. Addressing this during sign, a parameter 'k' is fixed with ko $\geq$ m, the UOV map $\mathcal{P}(x)$: $\mathbb{F}_q^n \rightarrow \mathbb{F}_q^m$ is whipped to $\mathcal{P}^*(x_1, x_2, ..x_k)$: $\mathbb{F}_q^{kn} \rightarrow \mathbb{F}_q^m$ using public matrices called emulsifier maps E_{ij} as

$$\mathcal{P}^*(x_1, \ldots, x_k) := \sum_{i=1}^{k} E_{ii} P(x_i) + \sum_{i=1}^{k} \sum_{j=i+1}^{k} E_{ij} P'(x_i, x_j)$$

With ko degrees of freedom, a solution is guaranteed to be found. The whipped map $\mathcal{P}^*$ is constructed in such a way that it vanishes on the subspace $O^k = (o_1, ..., o_k)$, $\forall$ i $\in$ k, $o_i \in \mathcal{O}$. The target t is computed as $\mathcal{H}(M||Salt)$. The signer at random chooses $(v_1, ..., v_k) \in \mathbb{F}_q^{kn}$ and solves for $(o_1, ..., o_k) \in O_k$ such that $\mathcal{P}^*(v_1 + o_1, ..., v_k + o_k) =$ t. The solution ($s_i = v_i + o_i$ where i $\in$ k) along with the salt is shared as signature. For verification, message M with the salt is hashed to obtain t and the signature is accepted if and only if $\mathcal{P}^*(s_i) =$ t where i $\in$ k

3 Implementation Details

The implementation follows largely the structure and implementation strategies of the Reference implementation of Mayo team, as included in the NIST submission package. Bitslicing in the Jasmin implementations follows as that of the C implementation. Bitslicing is done based on the parameter 'm', the number of equations in the Mayo2 specification.

Finite Field Implementation. At the bottom is the implementation of the GF(16) arithmetic. The field elements are obtained as $\mathbb{Z}_2[x]/(x^4+x+1)$. Bitsliced arithmetic for carrying out 64 field additions, multiplications is implemented. A simple finite field arithmetic using 64-bit registers in Jasmin without slicing is also implemented. The simple arithmetic is only used towards the end of the signature generation in putting together the obtained values of the oil and vinegar variables to construct the signature. The rest of the Mayo2 implementation uses the bitsliced arithmetic.

Linear Algebra Operations. This layer of implementation includes the matrix-matrix multiplication, addition, and matrix-vector multiplications for operating on large bitsliced matrices of finite field elements.

SHAKE256 Implementation. The Mayo specification indicates the use of SHAKE256 at various places of its design. This includes expanding the secret key seed bytes to the public key seed bytes and the bytes to generate the matrix O, hashing the message to the digest etc. SHAKE-256 jasmin implementation as available in the libjade library [1] has been modified to suit the requirement and has been used. Wrapper functions have been written in jasmin with the required input and output lengths of the SHAKE256 call.

PK-PRF Implementation. AES-128-CTR based PK PRF is used in the jasmin implementation as specified. These are used to expand the block matrices P_1 and P_2 in the public key. We have constructed the PK PRF in jasmin on top of the AES-128-CTR code available in the formosa-crypto website [2]. The counter initialised to zero is encrypted using AES-128 and are concatenated to fill the required buffer size. It can be noted that the AES-128-CTR implementation uses AES-NI instructions for its rounds.

Randombytes Generation. The random bytes for the initial seed_sk and the salt for randomising signatures are using the #randombytes primitive in Jasmin which derives random bytes from the linux CSPRNG /dev/urandom.

Keygen, Sign and Verify. Keygen and Verify modules have been implemented using bitsliced operations as in the C reference implementation.The compact secret key is expanded as specified. The OV map is whipped to a larger map using the emulsifier maps. Vinegar values are sampled in the whipped up map and are solved for the oil variables. The echelon form is implemented in a bitsliced fashion.

Benchmarking The Table 1 included in appendix A, denotes the average execution time computed over 100 runs of the implementation on a 2.25GHz x86_64 processor with 256 GB RAM. The benchmarked C Reference implementation is without AES-NI enabled in the NIST submission package. We plan to perform detailed benchmarking of the Jasmin implementation of Mayo2 against the C Reference implementation with and without AES-NI enabled over 1,00,000 runs.

4 Future Work

We plan to (i) formally verify the Jasmin implementation for functional correctness and prove functional equivalence to the C reference implementation using Easycrypt and Z3 (ii) formally verify the implementation for constant-time execution through logically reasoned machine-checked proofs using Easycrypt at the software level.

References

1. `https://github.com/formosa-crypto/libjade`, libjade
2. `https://formosa-crypto.org/news/2022-06-07/sibenik`, formosa Crypto
3. Almeida, J.B., et. al.: Machine-checked proofs for cryptographic standards: Indifferentiability of sponge and secure high-assurance implementations of sha-3. In: Proceedings of the 2019 ACM SIGSAC Conference (2019)
4. Almeida, J.B., Barbosa, M.: Jasmin: High-assurance and high-speed cryptography. In: Proceedings of the ACM SIGSAC Conference on Computer and Communications Security. pp. 1807–1823 (2017)
5. Almeida, J.B., Barbosa, M., Barthe, G.: Formally verifying kyber episode iv: Implementation correctness (2023)

6. Barthe, G., Dupressoir, F., Grégoire, B., Kunz, C., Schmidt, B., Strub, P.Y.: Easy-Crypt: A Tutorial, pp. 146–166. Springer International Publishing, Cham (2014)
7. Beullens, W.: Mayo: Practical post-quantum signatures from 0il-and-vinegar maps. In: International Conference on Selected Areas in Cryptography. pp. 355–376. Springer (2021)

A Appendix

The Mayo2 parameters, as elaborated in Section 2, utilized in our implementation are m = 64, n = 78, k = 4, o = 18 with a secret key size of 24B, public key size of 5488B and signature size of 180B.

Table 1. Execution time consumed for Jasmin and Reference Implementations in μsec.

Primitive	**Jasmin Impl.**	**C Reference Impl.**
Keygen	930	1629
Sign	3206	2749
Verify	480	665