Journal of The Korea Institute of Information Security & Cryptology ISSN 1598-3986(Print)
VOL.33, NO.2, Apr. 2023 ISSN 2288-2715(Online)
https://doi.org/10.13089/JKIISC.2023.33.2.243

Three-level 하이브리드 몽고메리 감산을 통한 ARM Cortex-M7에서의 CSIDH-512 최적화*

최 영 록,[1†] 허 동 회,[1] 홍 석 희,[2] 김 수 리[3‡]
[1,2]고려대학교 (대학원생, 교수), [3]성신여자대학교 (교수)

Optimized Implementation of CSIDH-512 through Three-Level Hybrid Montgomery Reduction on ARM Cortex-M7*

Younglok Choi,[1†] Donghoe Heo,[1] Seokhie Hong,[2] Suhri Kim[3‡]
[1,2]Korea University (Graduate student, Professor),
[3]Sungshin Women's University (Professor)

요 약

NIST PQC 표준화 작업 Round 4에 올라 있던 유일한 아이소제니 기반 KEM 알고리즘인 SIKE에 대한 효율적인 키 복구 공격이 발표됨에 따라, 이를 대체할 수 있는 키 교환 알고리즘인 CSIDH가 다시 주목받고 있다. CSIDH는 현재까지 알려진 공격에 안전한 아이소제니 기반 키 교환 알고리즘으로, CRS 스킴을 현대화하여 효율적인 NIKE를 제공한다. 본 논문에서는 CSIDH-512를 ARM Cortex-M7에서 구현하고 three-level 하이브리드 몽고메리 감산을 적용하여 최적화한 성능을 측정한 뒤 그 결과 및 한계에 대해 서술하고 향후 연구 방향을 제시한다. 이는 기존에 제시되지 않았던 32-bit 임베디드 기기에서의 CSIDH 구현으로, 향후 다양한 임베디드 환경에서 CSIDH 및 파생 암호 알고리즘을 구현하는데 본 논문의 결과를 이용할 수 있을 것으로 기대한다.

ABSTRACT

As an efficient key recovery attack on SIDH/SIKE was proposed, CSIDH is drawing attention again. CSIDH is an isogeny-based key exchange algorithm that is safe against known attacks to date, and provide efficient NIKE by modernizing CRS scheme. In this paper, we firstly present the optimized implementation of CSIDH-512 on ARM Cortex-M7. We use three-level hybrid Montgomery reduction and present the results of our implementation, limitations, and future research directions. This is a CSIDH implementation in 32-bit embedded devices that has not been previously presented, and it is expected that the results of this paper will be available to implement CSIDH and derived cryptographic algorithms in various embedded environments in the future.

Keywords: PQC, Key exchange, Isogeny, CSIDH, Montgomery reduction

I. 서 론

Shor가 1994년에 제안한 쇼어 알고리즘[1]은 다항 시간 내에 소인수분해 및 이산대수 문제를 해결할 수 있는 양자 알고리즘이다. 따라서 소인수분해, 이산대수 문제의 어려움에 안전성을 두고 있는 RSA, DSA, ECDSA 같은 공개키 암호화 및 전자서명 알고리즘들은 모두 양자 컴퓨팅 환경에서 쇼어 알고리

Received(03. 07. 2023), Accepted(03. 21. 2023)
* 이 성과는 2023년도 정부(과학기술정보통신부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임 (No.NRF-2020R1A2C1011769)
† 주저자, dudfhrchl@korea.ac.kr
‡ 교신저자, suhrikim@gmail.com(Corresponding author)

즘에 의해 다항 시간 내에 해결된다. 이러한 변화에 대응하기 위해 미국 국립표준기술연구소 (National Institute of Standards and Technology, NIST)는 양자 컴퓨팅 환경에서도 안전한 공개키 암호 알고리즘인 양자 내성 암호 (Post Quantum Cryptography, PQC)의 필요성을 제고하여 2017년 PQC 표준화 작업을 시작했고 현재 Round 4를 진행하고 있다.

양자 내성 암호는 기반 문제에 따라 여러 가지가 제시되고 있으며 대표적으로 격자 기반, 코드 기반, 아이소제니 기반, 해시 기반 암호 등이 있다. NIST는 PQC 표준화 작업을 통해 KEM (Key Encapsulation Mechanism)으로는 격자 기반 암호인 CRYSTALS-Kyber를, 전자서명 알고리즘으로는 격자 기반 암호인 CRYSTALS-Dilithium, Falcon과 해시 기반 암호인 SPHINCS$^+$를 표준 알고리즘으로 선정하고 코드 기반 암호인 BIKE, Classic McEliece, HQC와 아이소제니 기반 암호인 SIKE를 Round 4 후보로 두어 추가적인 표준화 작업을 진행하고자 한다.

SIKE (Supersingular Isogeny Key Encapsulation)[2]은 2011년에 Jao와 De Feo에 의해 제안된 키 교환 알고리즘인 SIDH (Supersingular Isogeny Diffie-Hellman)을 기반으로 하여 설계된 KEM이다. SIDH는 특유의 비가환적인 성질 때문에 키 교환에 필수적인 정보뿐만 아니라 추가적인 정보를 전달해야 하는데, 2022년 7월에 이러한 추가적인 정보를 이용한 효율적인 키 복구 공격 논문이 발표되어 더 이상 기존의 안전성을 유지할 수 없다고 평가된다[3].

CSIDH (Commutative SIDH)[4]는 Castryck 등에 의해 제안된 아이소제니 기반 키교환 알고리즘으로 CRS 스킴[5],[6]을 현대화하여 효율적인 NIKE (Non Interactive Key Exchange)를 제공한다. CSIDH는 SIDH와 달리 가환적인 성질을 가지고 있어 추가적인 정보를 전달하지 않기 때문에 기존의 안전성을 유지하고 있다. 공개키의 크기가 작다는 장점을 가진 CSIDH는 임베디드 기기를 포함한 경량환경에서의 통신환경에 적합하지만 경량환경에서의 CSIDH 구현 관련 논문은 빈약한 상황이다.

본 논문에서는 CSIDH-512를 ARM Cortex-M7을 마이크로컨트롤러로 가지는 STM32F769DISCOVERY 보드 위에 하이브리드 몽고메리 감산을 통해 최적 구현하여 성능을 측정하고 향후 연구 방향을 제시한다.

본 논문의 구성은 다음과 같다. 2장에서는 본 논문을 이해하는 데 필요한 CSIDH와 몽고메리 감산, ARM Cortex-M7 등의 배경지식에 관해 설명한다. 3장에서는 하이브리드 몽고메리 감산에 관해 설명하고 이를 통해 어떻게 ARM Cortex-M7에서 CSIDH를 최적화하였는지 설명한다. 4장에서는 구현 결과에 관해 설명한다. 마지막으로 5장에서는 결론 및 향후 연구 주제를 제시한다.

II. 배경지식

본 장에서는 CSIDH와 몽고메리 감산, ARM Cortex-M7을 이해하는데 필요한 배경지식을 소개한다.

2.1 CSIDH

CSIDH는 유한체 $\mathbb{F}_p$ 위에 정의된 supersingular 타원곡선에서 가환성을 가지는 group action을 이용한 아이소제니 기반 암호이다. O를 imaginary quadratic field에서의 order라 하고 $End_p(E)$는 $\mathbb{F}_p$ 위에서 정의된 supersingular 타원곡선의 endomorpghism ring이라 하자. 이때 $e\ell\ell_p(O)$를 다음과 같이 정의한다.

$$e\ell\ell_p(O) = \{E/\mathbb{F}_p \mid End_p(E) \equiv O\} \quad (1)$$

이 경우 O의 ideal class group $cl(O)$가 $e\ell\ell_p(O)$에서 자유롭고 전이적으로 작용한다는 사실은 잘 알려져 있다. 이 group action은 아래와 같이 표현할 수 있다.

$$\begin{array}{ccc} cl(O)\times e\ell\ell_p(O) & \rightarrow & e\ell\ell_p(O) \\ ([\iota],E) & \mapsto & [\iota]E \end{array} \quad (2)$$

여기서 $E \in e\ell\ell_p(O)$이고, $[\iota] \in cl(O)$이다.

CSIDH에서는 서로 다른 홀수인 작은 소수 $\ell_1,\ldots,\ell_n$에 대해서 소수 p를 $p = 4\ell_1\ell_2\cdots\ell_n - 1$로 정의하고 $End_p(E) = \mathbb{Z}[\pi]$를 만족하는 $\mathbb{F}_p$ 위에서 정의된 supersingular 타원곡선 E를 사용한다. 이때 π는 E의 Frobenious endomorphism을 의미한다. E가 $\mathbb{F}_p$ 위에서 정의된 supersingular 타원곡선이므로 π의 trace는 0이

되어 $\left|E(\mathbb{F}_p)\right| = p+1$을 만족한다. 이제 $O \cong End_p(E)$라 하면 각각의 소수 ℓ_i에 대해 $\pi^2 - 1 \equiv 0 \mod \ell_i$를 만족하기 때문에, ideal l_iO를 $l_iO = \iota_i\overline{\iota_i}$의 형태로 분해할 수 있고, 여기에서 $\iota_i = (\ell_i, \pi-1), \overline{\iota_i} = (\ell_i, \pi+1)$이다. 이 경우 group action $[\iota_i]E$는 Velu의 공식을 이용해서 $\mathbb{F}_p$ 위의 아이소제니 ϕ_{ι_i}로 연산할 수 있다.

Alice와 Bob이 서로 키를 교환한다고 하자. Alice는 개인키를 벡터 $(e_1,\cdots,e_n) \in \mathbb{Z}^n$로 선택하는데, 이때 각각의 정수 e_i는 양의 정수 m에 대해서 $e_i \in [-m, m]$의 범위를 가진다. 이 벡터를 통해 ideal class $[\iota] = [\iota_1^{e_1} \cdots \iota_n^{e_n}]$를 표현하여 group action $[\iota]E$을 계산한다. 이 연산은 실제로 $i = 1,\ldots,n$에 대해 ℓ_i차 아이소제니를 e_i번씩 계산하는 것으로 구성된다. Alice는 본인의 개인키 $[\alpha]$를 생성하여 group action을 통해 공개키 $E_A = [\alpha]E$를 계산하고 E_A를 Bob에게 전달한다. Bob도 Alice와 마찬가지로 개인키 $[\beta]$를 생성한 뒤, 공개키 $E_B = [\beta]E$를 계산하여 Alice에게 전달한다. Bob의 공개키를 받은 Alice는 E_B에 group action을 수행해 $[\alpha]E_B$를 연산하고, Bob도 마찬가지로 $[\beta]E_A$를 연산한다. 위 환경에서 사용하는 group action의 가환성에 의해 $[\alpha]E_B = [\beta]E_A$가 성립하고 이를 공유키로 가지며 Alice와 Bob의 키 교환은 종료된다.

2.2 몽고메리 감산

CSIDH의 연산 중 시간을 가장 많이 소비하는 연산은 유한체 곱셈이다. 따라서 본 논문에서는 몽고메리 감산을 통해 구현된 유한체 곱셈을 최적화하여 CSIDH를 최적화하였다.

몽고메리 감산은 1985년에 Montgomery에 의해 제안된 가장 유명한 모듈러 감산 기술로[7], 비효율적인 연산인 나눗셈을 간단한 시프트 연산을 통해 대체하여 효율적으로 모듈러 감산을 수행하는 알고리즘이다. 이 알고리즘은 크게 주어진 두 정수를 곱하는 곱셈 부분과 곱한 결과를 나누는 감산 부분으로 이루어져 있다. 자세한 과정은 Fig. 1.에 서술되어있다.

주어진 두 개의 정수 X, Y와 법 M에 대해서 곱셈 $P = X \cdot Y \bmod R$을 몽고메리 감산을 통해 계산하기에 앞서 X와 Y를 먼저 $X' = X \cdot R \bmod M$와 $Y' = Y \cdot R \bmod M$와 같이 몽고메리 도메인 위로 변환한다. 이는 몽고메리 도메인 위의 입력값으로 몽고메리 감산을 수행해야 결괏값이 몽고메리 도메인의 값으로 출력되기 때문이다. 또한 효율적인 계산을 위해 몽고메리 나머지 R은 2의 거듭제곱으로 선택하고 $M' = -M^{-1} \bmod R$은 사전 계산한다. 그 후 몽고메리 감산을 수행하기 위해 다음의 세 값을 계산한다.

$$T = X \cdot Y \tag{3}$$

$$Q = T \cdot M' \bmod R \tag{4}$$

$$Z = (T + Q \cdot M)/R \tag{5}$$

모든 계산을 마친 후 $Z \geq M$일 때 Z를 $Z - M$로 갱신하고 마지막으로 Z를 결괏값으로 출력한다.

Algorithm 1. Montgomery reduction

Input : An m-bit modulus M, Montgomery radix $R = 2^m$, two m-bit operands X and Y, and m-bit pre-computed constant $M' = -M^{-1} \bmod R$
Output : Montgomery product $(Z = (X \cdot Y) \cdot R^{-1} \bmod M)$

1. $T \leftarrow X \cdot Y$
2. $Q \leftarrow T \cdot M' \bmod R$
3. $Z \leftarrow (T + Q \cdot M)/R$
4. if $Z \geq M$ then $Z \leftarrow Z - M$ end if
5. return Z

Fig. 1. Montgomery reduction

2.3 ARM Cortex-M7

ARM Cortex-M7은 ARM Cortex-M 계열에서 최고 사양을 자랑하는 마이크로컨트롤러이다. 32-bit RISC 프로세서로 13개의 32-bit 범용 레지스터와 32개의 32-bit 부동소수점 레지스터를 제공하며 ARM Cortex-M4에 비해 약 두 배의 전력 효율성을 지니고 64-bit 부동소수점 레지스터를 추

가 제공한다. ARM Cortex-M7은 자동차, 산업 자동화, 의료 기기, 고급 오디오, 이미지 및 음성 처리, 센서 융합 및 모터 제어를 비롯한 다양한 분야에서 사용되고 있다.

본 논문에서는 ARM Cortex-M7에서 몽고메리 감산을 구현하기 위해 다음과 같은 64-bit 곱셈 명령어를 사용하였다.

```
UMLAL R0, R1, R2, R3
→ (R1,R0) = (R1,R0) + (R2×R3)          (6)
```

UMLAL은 레지스터 R0 (low 32-bit)와 R1 (high 32-bit)에 저장된 64-bit 값에 레지스터 R2와 R3에 저장된 두 32-bit 값을 곱한 64-bit 값을 더하는 명령어이다. 하지만 UMLAL은 덧셈에서 생기는 오버플로우를 무시한다는 단점이 있어 본 논문의 구현에서는 high 32-bit에 해당하는 레지스터 R1을 0으로 갱신하는 방식을 통해 오버플로우를 방지하여 구현했다.

III. 하이브리드 몽고메리 감산을 이용한 최적화

본 장에서는 CSIDH-512 최적화를 위해 사용한 하이브리드 몽고메리 감산에 관해 설명하고 이를 통해 CSIDH-512를 ARM Cortex-M7에서 어떻게 최적 구현하였는지 설명한다.

3.1 하이브리드 몽고메리 감산

하이브리드 몽고메리 감산은 몽고메리 감산을 부분 몽고메리 감산 과정과 부분 곱셈 과정으로 나눠 수행하는 알고리즘이다[8]. [8]에서는 최적화를 위해 카라추바 알고리즘[9]을 통해 부분 곱셈 과정을 수행하였으니 본 논문에서는 product-scanning 기반 곱셈을 통해 부분 곱셈 과정을 수행하였다. 또한 [8]에서는 one-level 및 two-level 하이브리드 몽고메리 감산을 제안하고 구현 결과를 제시하였으나, 본 논문에서는 [8]에서 제시된 기법을 확장하여 three-level 하이브리드 몽고메리 감산을 제안하고 이를 통해 몽고메리 감산을 구현하였다.

M와 Q를 각각 m-bit의 길이를 가진 법과 몫이라고 하자. 워드의 크기를 w-bit라 하면 워드의 개수 $n=\lceil m/w \rceil$에 따라 M과 Q, 그리고 두 m-bit 정수 X, Y의 곱셈인 중간 결과 $T=X\cdot Y$를 다음과 같이 나타낼 수 있다.

$$M=(M[n-1],\cdots,M[2],M[1],M[0]) \tag{7}$$

$$Q=(Q[n-1],\cdots,Q[2],Q[1],Q[0]) \tag{8}$$

$$T=(T[2n-1],\cdots,T[2],T[1],T[0]) \tag{9}$$

Level이 높아질수록 부분 몽고메리 감산 과정은 원래의 크기의 절반에 대한 부분 몽고메리 감산 과정과 부분 곱셈 과정을 통해 수행된다. 먼저 일반적인 곱셈을 통해 $Q=T\cdot M' \bmod R$를 계산한 후, 하이브리드 방식을 통해 $Q\cdot M$을 계산한다.

Three-level 하이브리드 몽고메리 감산의 경우 Fig. 2.와 같은 과정을 거쳐 계산된다. Fig. 2.는 $n=16$일 때의 three-level 하이브리드 몽고메리 감산 과정을 나타낸 그림으로 마름모의 각 점은 단일 워드 곱셈 $Q[i]\times M[j]$ $(i,j=0,\cdots,n-1)$를 나타낸다. 마름모의 가장 오른쪽에 있는 점이 가장 작은 지표 $i=j=0$에 해당하는 점 $Q[0]\times M[0]$을 나타내고 가장 왼쪽에 있는 점이 가장 큰 지표 $i=j=n-1$에 해당하는 점 $Q[n-1]\times M[n-1]$를 나타낸다. 같은 수직선 위에 있는 단일 워드 곱셈 $Q[i]\times M[j]$ $(i+j=k)$를 더한 값을 중간 결과 $T[k]$ $(k=0,\cdots,2n-1)$로 나타낸다. 빨간 점에서는 $T\cdot M' \bmod R$를 계산하여 $Q[i]$를 생성한 후, 단일 워드 곱셈 $Q[i]\times M[j]$를 수행하고 검은 점에서는 빨간 점에서 생성된 $Q[i]$로 단일 워드 곱셈 $Q[i]\times M[j]$를 수행한다. Fig. 2.에 있는 숫자는 계산의 순서를 나타내는데 크게 Q를 계산하는 부분 몽고메리 감산 과정(①,③,⑥,⑧,⑫,⑭,⑰,⑲)과 이렇게 계산된 Q를 통해 $Q\cdot M$을 계산하는 부분 곱셈 과정으로 나누어진다 (②,④,⑤,⑦,⑨,⑩,⑪,⑬,⑮,⑯,⑱,⑳,㉑,㉒). 또한 부분 곱셈 과정은 three-level 부분 곱셈 과정 (②,④,⑦,⑨,⑬,⑮,⑱,⑳), two-level 부분 곱셈 과정(⑤,⑩,⑯,㉑), one-level 부분 곱셈 과정 (⑪,㉒)의 세 가지로 나누어진다.

부분 몽고메리 감산 과정 (①,③,⑥,⑧,⑫,⑭,⑰,⑲)에서는 앞서 언급했듯이 $T[i]\times M' \bmod R$를 수행해 $Q[i]$ $(i=0,\cdots,n-1)$을 생성한 후에 $Q[i]\times M[j]$ $(j=0,\cdots,[n/8]-1)$를 계산한다. 부분 몽고메리 감산 과정은 한번 수행될 때 $[n/8]$개의 $Q[i]$를 생성하고 이 $Q[i]$를 통해 $Q[i]\times M[j]$

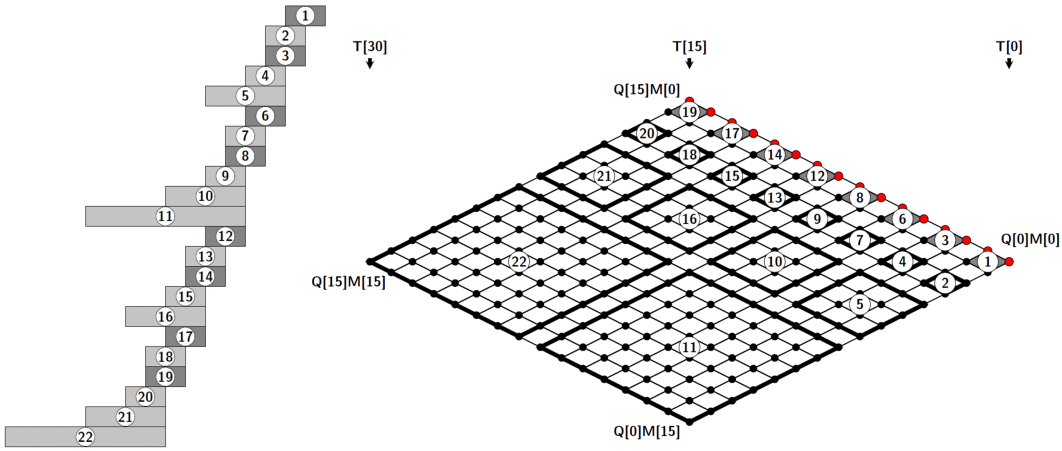

Fig. 2. Three-level hybrid Montgomery reduction ($n=16$)

$(j=0,\cdots,[n/8]-1)$를 계산하는데, 예를 들어 ①에서 $Q[i]$ $(i=0,1)$를 생성하고 이를 통해 $Q[i]\times M[j]$ $(j=0,1)$를 계산함을 볼 수 있다.

부분 몽고메리 감산 과정의 경우 일반적인 몽고메리 감산과 다르게 마지막에 조건부 뺄셈을 수행하지 않는다. 대신 Fig. 2.의 모든 과정을 수행하고 난 후에 조건부 뺄셈을 수행한다. 자세한 부분 몽고메리 감산 과정은 Fig. 3.에 서술되어있다.

과정 1과 과정 2의 경우 Fig. 1.의 몽고메리 감산과 같은 과정을 수행하지만 원래 계산에 쓰이는 인자의 1/8의 길이를 가지고 수행한다. 또한 앞서 서술하였듯이 조건부 뺄셈을 수행하지 않고 완전하지 않은 중간 결과 $\{CARRY,Z\}$와 몫 Q를 출력한다.

Algorithm 2. Sub-Montgomery reduction

Input : An $m/8$-bit 1/8 part of modulus M_p, where modulus M is m-bit, 1/8 part of Montgomery radix R_p, where Montgomery radix R is $R=2^m$, an operand T_p in the range $[0,2^{m/4})$, and pre-computed constant $M_p'=-M^{-1} \bmod R_p$
Output : Sub-Montgomery product $\{CARRY,Z\}=SubMonRed(T_p,M_p,R_p)$ $=T_p\cdot R_p^{-1} \bmod M_q$ and quotient Q (M_q is an $m/4$-bit 1/4 part of modulus)

1. $Q \leftarrow T_p \cdot M_p' \bmod R_p$
2. $\{CARRY,Z\} \leftarrow (T_p+Q\cdot M_p)/R_p$
3. **return** $\{CARRY,Z\},Q$

Fig. 3. Sub-Montgomery reduction of three-level hybrid Montgomery reduction

부분 곱셈 과정은 three-level 부분 곱셈 과정(②,④,⑦,⑨,⑬,⑮,⑱,⑳), two-level 부분 곱셈 과정(⑤,⑩,⑯,㉑), one-level 부분 곱셈 과정 (⑪, ㉒)으로 나누어지는데 부분 몽고메리 감산 과정과 달리 몫 Q 생성을 수행하지 않고 단순한 곱셈을 수행한다. Three-level 부분 곱셈 과정은 부분 하이브리드 감산 과정에서 생성한 $Q[i]$ $(i=0,\cdots,n-1)$와 $M[j]$ $(j=[n/8],\cdots,[n/4]-1)$를 곱해 $Q[i]\times M[j]$를 계산한다. Three-level 부분 곱셈 과정은 한번 수행될 때 직전에 수행된 부분 몽고메리 감산 과정에서 생성된 $[n/8]$개의 $Q[i]$를 통해 $Q[i]\times M[j]$ $(j=[n/8],\cdots,[n/4]-1)$를 계산하는데, 예를 들어 ②에서는 ①에서 생성된 $Q[i]$ $(i=0,1)$와 $M[j]$ $(j=2,3)$를 곱해 $Q[i]\times M[j]$를 계산하는 것을 확인할 수 있다.

부분 몽고메리 감산 과정과 three-level 부분 곱셈 과정을 두 번씩 수행하면 이는 two-level 하이브리드 감산의 부분 몽고메리 감산 과정을 한 번 수행한 것과 같다. ①~④의 과정을 수행하면 $Q[i]$ $(i=0,\cdots,3)$을 생성하고 이를 통해 $Q[i]\times M[j]$ $(j=0,\cdots,3)$을 계산했으므로 two-level 하이브리드 감산의 부분 몽고메리 감산 과정과 같은 결과를 출력함을 확인할 수 있다. 이처럼 부분 몽고메리 감산 과정을 낮은 level의 부분 몽고메리 감산 과정과 부분 곱셈 과정을 통해 수행하는 것이 high-level 하이브리드 몽고메리 감산의 특징이다. Two-level 부분 곱셈 과정은 three-level 부분 곱셈 과정과 같은 과정을 거치지만 계산하는 단일 워드 곱셈의 수는 4배이다. 직전에 두 번 수행된 부분 몽고메리 감산 과정

Algorithm 3. Three-level hybrid Montgomery reduction

Input : An m-bit modulus M, Montgomery radix $R=2^m$, and its 1/8 part radix $R_p=2^{m/8}$, an operand T, where $T=X\cdot Y$ in the range $[0,2^{2m})$

Output : Montgomery product $Z=MonRed(T,R,M)=T\cdot R^{-1} \bmod M$

1. $\{CARRY_1, Z_1\}, Q_1 \leftarrow$ $SubMonRed(T[0,2^{m/4}], M[0,2^{m/8}], R_p)$
2. $K_1 \leftarrow Q_1 \times M[2^{m/8}, 2^{m/4})$
3. $\{CARRY_2, K_1\} \leftarrow$ $K_1 + Z_1 + (T[2^{m/4}, 2^{m/4+m/8}) + CARRY_1)\cdot 2^{m/8}$
4. $\{CARRY_3, Z_2\}, Q_2 \leftarrow SubMonRed(K_1, M[0,2^{m/8}], R_p)$
5. $CARRY_3 \leftarrow CARRY_2 + CARRY_3$
6. $K_2 \leftarrow Q_2 \times M[2^{m/8}, 2^{m/4})$
7. $\{CARRY_4, K_2\} \leftarrow$ $K_2 + Z_2 + (T[2^{m/4+m/8}, 2^{m/2}) + CARRY_3)\cdot 2^{m/8}$
8. $K_3 \leftarrow (Q_2\cdot 2^{m/8} + Q_1)\times M[2^{m/4}, 2^{m/2})$
9. $\{CARRY_5, K_3\} \leftarrow$ $K_3 + K_2 + (T[2^{m/2}, 2^{m/2+m/4}) + CARRY_4)\cdot 2^{m/4}$
10. $\{CARRY_6, Z_3\}, Q_3 \leftarrow$ $SubMonRed(K_3[0,2^{m/4}], M[0,2^{m/8}], R_p)$
11. $K_4 \leftarrow Q_3 \times M[2^{m/8}, 2^{m/4})$
12. $\{CARRY_7, K_4\} \leftarrow$ $K_4 + Z_3 + (K_3[2^{m/4}, 2^{m/4+m/8}) + CARRY_6)\cdot 2^{m/8}$
13. $\{CARRY_8, Z_4\}, Q_4 \leftarrow$ $SubMonRed(K_4, M[0,2^{m/8}], R_p)$
14. $CARRY_8 \leftarrow CARRY_7 + CARRY_8$
15. $K_5 \leftarrow Q_4 \times M[2^{m/8}, 2^{m/4})$
16. $\{CARRY_9, K_5\} \leftarrow$ $K_5 + Z_4 + (K_3[2^{m/4+m/8}, 2^{m/2}) + CARRY_8)\cdot 2^{m/8}$
17. $CARRY_9 \leftarrow CARRY_5 + CARRY_9$
18. $K_6 \leftarrow (Q_4\cdot 2^{m/8} + Q_3)\times M[2^{m/4}, 2^{m/2})$
19. $\{CARRY_{10}, K_6\} \leftarrow$ $K_6 + K_5 + (T[2^{m/2+m/4}, 2^m) + CARRY_9)\cdot 2^{m/4}$
20. $K_7 \leftarrow (Q_4\cdot 2^{m/4+m/8} + Q_3\cdot 2^{m/4} + Q_2\cdot 2^{m/8} + Q_1)$ $\times M[2^{m/2}, 2^m)$
21. $\{CARRY_{11}, K_7\} \leftarrow$ $K_7 + K_6 + (T[2^m, 2^{m+m/2}) + CARRY_{10})\cdot 2^{m/2}$
22. $\{CARRY_{12}, Z_5\}, Q_5 \leftarrow$ $SubMonRed(K_7[0,2^{m/4}], M[0,2^{m/8}], R_{p)}$
23. $K_8 \leftarrow Q_5 \times M[2^{m/8}, 2^{m/4})$
24. $\{CARRY_{13}, K_8\} \leftarrow$ $K_8 + Z_5 + (K_7[2^{m/4}, 2^{m/4+m/8}) + CARRY_{12})\cdot 2^{m/8}$
25. $\{CARRY_{14}, Z_6\}, Q_6 \leftarrow SubMonRed(K_8, M[0,2^{m/8}], R_p)$
26. $CARRY_{14} \leftarrow CARRY_{13} + CARRY_{14}$
27. $K_9 \leftarrow Q_6 \times M[2^{m/8}, 2^{m/4})$
28. $\{CARRY_{15}, K_9\} \leftarrow$ $K_9 + Z_6 + (K_7[2^{m/4+m/8}, 2^{m/2}) + CARRY_{14})\cdot 2^{m/8}$
29. $K_{10} \leftarrow (Q_6\cdot 2^{m/8} + Q_5)\times M[2^{m/4}, 2^{m/2})$
30. $\{CARRY_{16}, K_{10}\} \leftarrow$ $K_{10} + K_9 + (K_7[2^{m/2}, 2^{m/2+m/4}) + CARRY_{15})\cdot 2^{m/4}$
31. $\{CARRY_{17}, Z_7\}, Q_7 \leftarrow$ $SubMonRed(K_{10}[0,2^{m/4}], M[0,2^{m/8}], R_p)$
32. $K_{11} \leftarrow Q_7 \times M[2^{m/8}, 2^{m/4})$
33. $\{CARRY_{18}, K_{11}\} \leftarrow$ $K_{11} + Z_7 + (K_{10}[2^{m/4}, 2^{m/4+m/8}) + CARRY_{17})\cdot 2^{m/8}$
34. $\{CARRY_{19}, Z_8\}, Q_8 \leftarrow SubMonRed(K_{11}, M[0,2^{m/8}], R_p)$
35. $CARRY_{19} \leftarrow CARRY_{18} + CARRY_{19}$
36. $K_{12} \leftarrow Q_8 \times M[2^{m/8}, 2^{m/4})$
37. $\{CARRY_{20}, K_{12}\} \leftarrow$ $K_{12} + Z_8 + (K_{10}[2^{m/4+m/8}, 2^{m/2}) + CARRY_{19})\cdot 2^{m/8}$
38. $CARRY_{20} \leftarrow CARRY_{16} + CARRY_{20}$
39. $K_{13} \leftarrow (Q_8\cdot 2^{m/8} + Q_7)\times M[2^{m/4}, 2^{m/2})$
40. $\{CARRY_{21}, K_{13}\} \leftarrow$ $K_{13} + K_{12} + (K_7[2^{m/2+m/4}, 2^m) + CARRY_{20})\cdot 2^{m/4}$
41. $CARRY_{21} \leftarrow CARRY_{11} + CARRY_{21}$
42. $K_{14} \leftarrow (Q_8\cdot 2^{m/4+m/8} + Q_7\cdot 2^{m/4} + Q_6\cdot 2^{m/8} + Q_5)$ $\times M[2^{m/2}, 2^m)$
43. $\{CARRY_{22}, K_{14}\} \leftarrow$ $K_{14} + K_{13} + (T[2^{m+m/2}, 2^{2m}) + CARRY_{21})\cdot 2^{m/2}$
44. if $\{CARRY_{22}, K_{14}\} \geqq M$ then $K_{14} \leftarrow \{CARRY_{22}, K_{14}\} - M$ end if
45. return K_{14}

Fig. 4. Three-level hybrid Montgomery reduction

에서 생성된 $[n/4]$개의 $Q[i]$를 통해 $Q[i]\times M[j]$ $(j=[n/4],\cdots,[n/2]-1)$를 계산하는데, 예를 들어 ⑤에서는 ①과 ③에서 생성된 $Q[i]$ $(i=0,\cdots,3)$와 $M[j](j=4,\cdots,7)$를 곱해 $Q[i]\times M[j]$를 계산함을 알 수 있다. ①~⑩의 과정을 수행하면 $Q[i]$ $(i=0,\cdots,7)$을 생성하고 이를 통해 $Q[i]\times M[j]$ $(j=0,\cdots,7)$을 계산했으므로 one-level 하이브리드 몽고메리 감산의 부분 몽고메리 감산 과정과 같은 결과를 출력함을 확인할 수 있다. One-level 부분 곱셈 과정 또한 같은 과정을 거쳐 two-level 부분 곱셈 과정의 4배의 단일 워드 곱셈을 계산한다. 직전에 4번 수행된 부분 몽고메리 감산 과정에서 생성된 $[n/2]$개의 $Q[i]$를 통해 $Q[i]\times M[j]$ $(j=[n/2],\cdots,n-1)$를 계산하는데, 예를 들어 ⑪에서는 ①,③,⑥,⑧에서 생성된 $Q[i]$ $(i=0,\cdots,8)$와 $M[j]$ $(j=8,\cdots,15)$를 곱해 $Q[i]\times M[j]$를 계산하는 것을 확인할 수 있다. ①~⑪의 과정을 반복해서 ⑫~㉒에서 한번 더 수행하고 마지막으로 조건부 뺄셈을 수행하면 three-level 하이브리드 몽고메리 감산은 마무리된다. 자세한 과정은 Fig. 4.에 서술되어있다.

3.2 ARM Cortex-M7에서의 구현

CSIDH-512는 최대 512-bit 소수의 유한체를 사용한다. 따라서 three-level 부분 몽고메리 감산 과정의 크기는 64-bit에 해당하므로 ARM Cortex-M7의 레지스터로 추가적인 load와 store를 수행하지 않고 구현할 수 있다. 자세한 과정은 Fig. 5.에 서술되어있고 관련 레지스터 스케줄링은 Table. 1.에 서술되어있다.

ARM-v7에서는 zero register가 존재하지 않기 때문에 R4를 0으로 초기화시켜 캐리를 계산할 때 zero register로 사용했다. $Q_i\times M_j$의 경우 덧셈에서 생기는 오버플로우를 없애기 위해 과정 8, 19처

Table 1. Register scheduling of 64-bit sub-Montgomery reduction on ARM Cortex-M7

Register	Operand
R0-R3	pointer of $T, Q, M, Carry$
R4	zero register
R5-R6	M
R7-R9	T
R10-R11	Q
R12	M'

Algorithm 4. 64-bit sub-Montgomery reduction on ARM Cortex-M7

Input : Intermediate results pointer R0 of T, quotient pointer R1 of Q, modulus pointer R2 of M, carry pointer R3 of $Carry$, constant pointer Mprime of M',
Output: Intermediate results T_2-T_3, quotient Q_0-Q_1, carry $Carry$

1.	MOV	R4, #0	
2.	LDR	R5, [R2]	$[M_0]$
3.	LDR	R6, [R2, 4]	$[M_1]$
4.	LDR	R7, [R0]	$[T_0]$
5.	LDR	R8, [R0, 4]	$[T_1]$
6.	LDR	R12, Mprime	$[M']$
7.	MUL	R10, R7, R12	$[Q_0=T_0\times M']$
8.	MOV	R9, #0	
9.	UMLAL	R7, R9, R10, R5	$[T_0+Q_0\times M_0]$
10.	ADDS	R8, R8, R9	$[T_1T_0+Q_0\times M_0]$
11.	LDR	R9, [R0, 8]	$[T_2]$
12.	ADC	R9, R9, R4	
13.	UMLAL	R8, R7, R10, R6	$[T_1+Q_0\times M_1]$
14.	ADDS	R9, R9, R7	$[T_2T_1+Q_0\times M_1]$
15.	LDR	R7, [R0, 12]	$[T_3]$
16.	ADC	R7, R7, R4	
17.	STR	R10, [R1]	$[Q_0]$
18.	MUL	R11, R8, R12	$[Q_1=T_1\times M']$
19.	MOV	R10, #0	
20.	UMLAL	R8, R10, R11, R5	$[T_1+Q_1\times M_0]$
21.	ADDS	R9, R9, R10	$[T_2T_1+Q_1\times M_0]$
22.	ADC	R7, R7, R4	
23.	UMLAL	R9, R8, R11, R6	$[T_2+Q_1\times M_1]$
24.	ADDS	R7, R7, R8	$[T_3T_2+Q_1\times M_1]$
25.	ADC	R8, R4, R4	$[Carry]$
26.	STR	R9, [R0, 8]	$[T_2]$
27.	STR	R7, [R0, 12]	$[T_3]$
28.	STR	R8, [R3]	$[Carry]$
29.	STR	R11, [R1, 4]	$[Q_1]$

Fig. 5. 64-bit sub-Montgomery reduction on ARM Cortex-M7

Table 2. Comparison with Montgomery reduction and three-level hybrid Montgomery reduction

Implementation	Multiplication	Reduction
Montgomery reduction	Product-scanning	Product scanning
Three-level hybrid Montgomery reduction	Product-scanning	Three-level hybrid Montgomery reduction

럼 상위 32-bit를 MOV 명령어를 통해 0으로 초기화시킨 후 UMLAL 명령어를 사용하는 방식으로 구현했는데 $Q_0 \times T_1$와 $Q_1 \times T_1$의 경우 앞선 과정에서 각각 레지스터 R7와 R8이 0으로 초기화되었기 때문에 추가적인 초기화 과정을 거치지 않고 구현했다.

IV. 구현 결과

본문에서는 일반적인 몽고메리 감산을 통한 구현과 three-level 하이브리드 몽고메리 감산을 통한 구현을 비교하고 실제 측정 결과를 비교한다. 이 때 CSIDH 구현에 사용한 초기 타원 곡선은 $y^2 = x^3 + x$이다.

4.1 몽고메리 감산과의 비교

몽고메리 감산과 three-level 하이브리드 몽고메리 감산을 비교하기 위해 일반적인 몽고메리 감산 또한 ARM Cortex-M7에 구현하여 비교하였다. Table. 2.는 본 논문에서 구현한 몽고메리 감산과 three-level 하이브리드 몽고메리 감산을 비교한 표이다. 동일한 조건에서의 비교를 위해 두 구현 모두 곱셈은 동일한 product-scanning 기반 곱셈을 사용했고 일반적인 몽고메리 감산의 감산 과정도 곱셈과 동일한 product-scanning 기반 곱셈을 사용하여 구현하였다. Table. 3.은 일반적인 몽고메리 감산과 three-level 하이브리드 몽고메리 감산의 load, store 명령어 사용횟수를 비교한 표이다. store 명령어 사용횟수는 29번 증가했지만, load 명령어 사용횟수가 327번 감소해 메모리 접근 명령어 사용횟수가 약 24% 감소한 것을 볼 수 있다.

Table 3. Comparison of the number of load and store instructions in ordinary and three-level hybrid Montgomery reduction implementation

	Ordinary	Three-level hybrid
load	1,140	813
store	112	141

4.2 실험 환경 및 측정 결과

ARM Cortex-M7을 마이크로컨트롤러로 탑재한 STM32F32F769IDISCOVERY를 타겟보드로 사용해 측정했다. 실험에 사용한 통합 개발 환경(Intergrated Development Environment, IDE)은 ST사에서 제공하는 STM32CubeIDE이다. 운영체제 없이 보드를 동작하므로 ST사에서 제공하는 STM32CubeMX를 통해 운영체제 역할을 해줄 HAL(Hardware Abstraction layer) 라이브러리를 생성해 레지스터를 세팅한 후 측정을 진행했다.

Table. 4.는 각 몽고메리 감산에 대해 CSIDH-512의 성능을 측정한 표로 단위는 사이클이다. CSIDH는 키 교환을 1번 수행할 때 키 생성과 공유키 계산에서 각각 2번의 group action을 수행하므로 총 4번의 group action을 수행한다. 따라서 50번의 키 교환을 수행해 200번의 group action의 평균, 최소, 최대 사이클 수를 측정했다. 시스템 클럭은 24MHz, 최적화 옵션은 -O3로 설정했다.

평균, 최소, 최댓값 모두 three-level 하이브리드 몽고메리 감산을 통해 group action을 수행하는 사이클이 일반적인 몽고메리 감산을 통해 group action을 수행하는 사이클의 약 75% 수준임을 알 수 있다. 즉 속도 측면에서 three-level 하이브리드 몽고메리 감산이 일반적인 몽고메리 감산보다 약 1.33배의 성능 개선이 있음을 확인했다.

Table 4. Performance results of group action of CSIDH-512 (in clock cycle)

	Ordinary	Three-level hybrid
Avg.	2,594,043,398	1,953,742,613
Min.	2,131,776,798	1,571,062,416
Max.	3,120,140,202	2,373,861,253

V. 결 론

본 논문에서는 NIST PQC 보안 강도 1에 해당하

는 CSIDH-512를 ARM Cortex-M7를 마이크로컨트롤러로 탑재한 STM32F769IDISCOVERY 보드 위에서 [8]에서 제시한 기법을 확장한 three-level 하이브리드 몽고메리 감산을 통해 최적화하여 성능 측정을 진행하였다.

구현한 three-level 하이브리드 몽고메리 감산은 일반적인 몽고메리 감산보다 store 명령어 사용횟수가 29번 증가하고 load 명령어 사용횟수가 327번 감소해 메모리 접근 명령어 사용횟수가 24% 감소하였다. 또한 실제 측정에서 일반적인 몽고메리 감산보다 약 75%의 사이클 만에 CSIDH-512의 group action을 수행함을 확인했다.

본 논문에서는 일반적인 몽고메리 감산과 three-level 하이브리드 몽고메리 감산의 단적인 비교를 위해 곱셈 과정을 product-scanning으로 고정했는데 향후 연구 과제로 곱셈 과정에 여러 곱셈 최적화 적용을 통한 몽고메리 감산 최적화를 진행할 것이다. 또한 타겟 환경에서의 최적화 기법 및 CSIDH 알고리즘의 최적화 기법을 적용하여 더욱 향상된 성능을 보일 수 있도록 할 것이다. 이러한 연구를 통해 작은 키 사이즈를 지닌 장점을 살려 CSIDH 및 이를 기반으로 한 전자서명 알고리즘인 CSI-FiSh 등의 알고리즘들을 다양한 임베디드 기기에서 활용할 수 있는 방안에 대한 논의를 활발히 이어 나갈 수 있을 것으로 기대한다.

References

[1] P.W. Shor, "Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer," SIAM Journal Computing, vol. 26, no. 5, pp. 1484-1509, Oct. 1997.

[2] R. Azarderakhsh et al., "Supersingular isogeny key encapsulation," submission to the NIST post-quantum standardization project, Nov. 2017.

[3] W. Castryck and T. Decru, "An efficient key recovery attack on SIDH (preliminary version)", IACR ePrint 2022-975, Jul. 2022.

[4] W. Castryck, et al., "CSIDH: an efficient post-quantum commutative group action," Advances in Cryptology - ASIACRYPT 2018, ASIACRYPT 2018, pp. 395-427, Dec. 2018.

[5] J.M. Couveignes, "Hard homogeneous spaces,", IACR ePrint 2006-291, Aug. 2006.

[6] A. Stolbunov, "Constructing public-key cryptographic schemes based on class group action on a set of isogenous elliptic curves," Advances in Mathematics of Communications, vol. 4, no. 2, pp. 215-235, Apr. 2010.

[7] P.L. Montgomery, "Modular multiplication without trial division," Mathematics of computation, vol. 44, no. 170, pp. 519-521, Apr. 1985.

[8] H. Seo, et al., "Hybrid Montgomery reduction," ACM Transactions on Embedded Computing Systems, vol. 15, no. 3, pp. 1-13, May. 2016.

[9] A.A. Karatsuba and Yu. Ofman, "Multiplication of many-digital numbers by automatic computers," Doklady Akademii Nauk, vol. 145, no. 2, pp. 293-294, Sep. 1962.

〈저자소개〉

최 영 록 (Younglok Choi) 정회원
2021년 2월: 고려대학교 수학과 졸업
2021년 3월~현재: 고려대학교 정보보호대학원 석사과정
〈관심분야〉 후양자암호

허 동 회 (Donghoe Heo) 학생회원
2019년 2월: 한양대학교 수학과 졸업
2019년 3월~현재: 고려대학교 정보보호대학원 석박사 통합과정
〈관심분야〉 후양자암호

홍 석 희 (Seokhie Hong) 종신회원
1995년: 고려대학교 수학과 학사
1997년: 고려대학교 수학과 석사
2001년: 고려대학교 수학과 박사
1999년 8월~2004년 2월: ㈜시큐리티 테크놀로지 선임연구원
2003년 3월~2004년 2월: 고려대학교 정보보호기술연구센터 선임연구원
2004년 4월~2005년 2월: K.U. Leuven ESAT/SCD-COSIC 박사후연구원
2005년 3월~2013년 8월: 고려대학교 정보보호대학원 부교수
2013년 9월~현재: 고려대학교 정보보호대학원 정교수
〈관심분야〉 대칭키 및 공개키 암호 알고리즘, 부채널 공격 및 대응기법, 디지털포렌식

김 수 리 (Suhri Kim) 정회원
2014년 2월: 고려대학교 수학과 이학사
2016년 8월: 고려대학교 정보보호학과 공학석사
2020년 2월: 고려대학교 정보보호대학원 공학박사
2020년 3월~2021년 2월: 고려대학교 정보보호대학원 박사후연구원
2020년 3월~2021년 2월: K.U. Leuven ESAT/SCD-COSIC 박사후 연구원
2022년 3월~현재: 성신여자대학교 수리통계데이터사이언스학부 조교수
〈관심분야〉 공개키 암호시스템, 후양자암호