정 보 보 호 학 회 지
제34권 제2호, 2024. 4

MPCitH 기반 영지식 증명과 대칭키 프리미티브 기반 일방향 함수를 결합 양자 내성 전자서명 AIMer 소개

하 진 철*, 김 성 광**, 손 민 철*

요 약

양자 컴퓨팅 환경을 대비한 암호 시스템에 대한 중요성이 대두되면서 국내외로 양자 내성 암호 시스템 표준 알고리즘을 선정하려는 노력이 이루어지고 있다. AIMer는 MPCitH 기반 영지식 증명 시스템인 BN++을 개선하고 대칭키 프리미티브 기반의 일반향 함수 AIM2를 결합하여 만들어진 전자서명으로, 현재 NIST의 양자 내성 전자서명 추가 라운드 및 양자 내성 암호 국가 공모전(KpqC) 2라운드를 진행 중인 알고리즘이다. 많은 양자 내성 암호 시스템이 격자 문제에 안전성을 기반하고 있는 것에 비해 AIMer 전자서명은 사용하는 대칭키 프리미티브 기반 일방향 함수 AIM2의 일방향성에 안전성을 기반하고 있으며, 성능적인 측면에서도 현재까지 선정된 NIST 표준 알고리즘 및 KpqC 2라운드 후보 알고리즘 내에서는 비격자 문제 기반 전자서명 중 공개키 크기와 서명 크기를 합쳤을 때 가장 작은 크기를 가지고 있다.

Ⅰ. 서 론

양자 컴퓨팅에 대한 많은 연구가 이루어지고, 기존에 사용하던 많은 암호 체계들이 양자 컴퓨팅 환경 상에서 안전하지 않다는 것이 밝혀지면서 양자 내성을 가지는 암호 체계에 대한 연구의 중요성이 커지고 있다. NIST에서는 2016년 양자 내성 암호 표준화 공모전을 시작하여 최근 표준 알고리즘이 선정되었으며, 한국에서도 양자 내성 공개키 암호 및 전자서명 표준화 공모전이 진행 중이다.

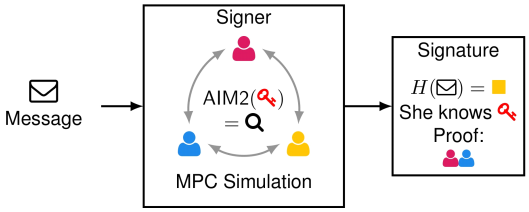
NIST 표준화 공모전에서 현재까지 선정된 전자서명 알고리즘은 총 3종류로, 격자 문제 안전성에 기반한 CRYSTAL-Dilithium[1] 및 Kyber[2]와 해시 함수 안전성에 기반한 SPHINCS+[3]이다. 격자 문제에 기반한 전자서명은 우수한 성능을 보여주지만, NIST는 격자 문제 외에 다양한 안전성에 기반한 전자서명을 고려하기 위해 추가적인 전자서명을 후보로 받아 추가 라운드를 진행하고 있다.

AIMer 전자서명은 이러한 요구에 부합하여 격자 기반 문제가 아닌, 영지식 증명 프로토콜과 대칭키 프리미티브 기반 일방향 함수를 결합하여 만든 전자서명이다. 일반적으로 영지식 증명 프로토콜과 일방향 함수를 결합하면 전자서명을 설계할 수 있는데, 주어진 일방향 함수 f의 입출력 쌍 (x,y)가 있을 때, x를 비밀키, y를 공개키로 설정한 다음, $f(x)=y$를 만족하는 비밀키 x를 알고 있다는 사실을 x를 드러내지 않고 증명함으로써 전자서명을 만들 수 있다. 영지식 증명 프로토콜의 안전성이 수학적으로 증명되어 있을 때, 전자서명의 안전성은 사용하는 함수의 일방향성에만 의존한다. AIMer는 대칭키 프리미티브에 기반한 일방향 함수를 설계하고, 이 회로를 효율적으로 계산할 수 있는 영지식 증명 프로토콜을 결합하여 만들어진 전자서명이다.

AIMer의 영지식 증명 프로토콜은 MPC-in-the-Head 패러다임에 기반하고 있다. MPCitH 패러다임은 Ishai 등이 제안한 방식으로, 다자간 연산(MPC) 프로토콜을 시뮬레이션하여 영지식 증명 프로토콜을 설계하는 방식이다[4]. AIMer 버전 1.0에서는 BN++ 영지식 증명 시스템[5]을 사용하였으며, 버전 2.0에서는 이를 AIMer에 맞춰 개선시켜서 성능을 높였다.

사용하는 내부 일방향 함수로는 버전 1.0에서는 대칭키 프리미티브에 기반하여 설계된 함수 AIM[6]을 사용하였고, 이후 다양한 제3자 분석이 제안됨에 따라[7,8] 버전 2.0에서는 효율성을 유지하면서 안전성

* 한국과학기술원 정보보호대학원 (대학원생, {smilecjf, encrypted.def}@kaist.ac.kr)
** 삼성 SDS (연구원, sk39.kim@samsung.com)



〔그림 1〕 AIMer 전자서명의 구조

을 높인 AIM2[9]로 업데이트 되었다. 그림 1에 AIMer 전자서명의 구조가 나타나있다.

AIMer 전자서명은 MPCitH 패러다임과 대칭키 프리미티브를 결합한 전자서명의 공통적인 특징으로 작은 비밀키, 공개키 크기를 가진다. 일반적으로 격자 기반 문제에 기반한 전자서명보다는 서명의 크기가 크다는 단점이 있지만, AIMer는 영지식 증명 프로토콜의 개선 및 효율적인 일방향 함수 설계를 통해 서명의 크기를 줄이는 최적화를 진행하였다. 그 결과 NIST 선정 알고리즘 및 KpqC 2라운드 후보 알고리즘 중 비격자 문제 기반 카테고리에서 공개키 크기 및 서명 크기를 합쳤을 때 가장 작은 크기를 가진다.

Ⅱ. 배경 지식

2.1. MPC-in-the-Head 패러다임

MPC-in-the-Head(MPCitH) 패러다임은 Ishai 등이 제안한 다자간 연산(MPC) 프로토콜 기반 영지식 증명 프로토콜 설계 방식이다[4]. 증명자(prover)가 함수 f와 입력 x에 대해 $y=f(x)$를 공개한 후 $f(x)=y$가 되는 x를 알고 있음을 증명하려고 할 때, x로부터 $f(x)$를 계산하는 MPC 프로토콜을 가상으로 수행한다. $f(x)$를 계산하는 MPC 프로토콜이 증명자의 머리 속에서 완료되면, 검증자는 임의로 k명의 참여자를 선택한다. 증명자는 선택된 k명의 참여자의 내부 관점에서의 정보를 검증자에게 공개한다. 증명자가 사용한 MPC 프로토콜에서는 k명의 참여자 만으로는 얻을 수 있는 정보가 없기 때문에, 검증자가 입력 x에 대한 정보를 알아낼 수는 없다.

검증자는 증명자가 $f(x)=y$가 되는 올바른 입력 x를 이용하여 MPC 프로토콜을 수행한 것이 맞는지 확인한다. 공개된 참여자의 내부 관점 정보를 바탕으로 프로토콜이 올바르게 수행되었는지, 중간에 커밋한 값이 올바른지 확인한다. 만약 악의적인 증명자가 $f(x)=y$가 되는 x의 값을 모르는 채로 프로토콜을 수행할 경우 검증자가 선택하지 않은 $N-k$명의 참여자에 대해서만 잘못된 계산을 수행했다면 검증자를 속일 수 있다. 공개할 k명의 참여자는 검증자가 선택하는 것이므로 악의적인 증명자가 검증을 통과할 확률은 $(N-k)/N$이다.

MPCitH 패러다임은 기본적으로 MPC 프로토콜을 가상으로 수행하여 영지식 증명을 하려는 것이므로 회로 계산이 아닌 회로 검증 형태로 적용이 가능하다. 만약 $x\neq 0$에 대해 $y=x^{-1}$를 계산한다고 할 때, MPC 프로토콜을 따라 x로부터 y를 계산하는 과정은 비효율적이지만 MPCitH 환경에서는 증명자가 올바른 y에 대한 share를 미리 계산하여 참여자에게 분배한 후 $xy=1$을 검증하는 과정으로 대체할 수 있다.

2.2. BN++ 영지식 증명 시스템

BN++ 증명 시스템은 Kales와 Zaverucha가 제안한 영지식 증명 시스템으로, 크기가 큰 유한체 상에서의 다항 연산을 효율적으로 검증할 수 있다[5]. Baum과 Nof가 제안한 BN 증명 시스템[10]을 개선한 것으로, 덧셈 기반의 share를 사용하여 선형 연산은 참여자들이 내부적으로 계산하고 곱셈 연산은 곱셈 검증 프로토콜을 이용하여 검증한다.

2.2.1. 곱셈 검증 프로토콜

유한체 $\mathbb{F}$ 상에서 C번의 곱셈 $z_j=x_j\cdot y_j$, $j=1,...,C$를 검증하는 과정은 다음과 같다. 추가적으로 사전 곱셈 쌍 $((a_j,b_j)_{j=1}^{C},c)\in(\mathbb{F}^2)^C\times\mathbb{F}$이 필요한데, $b_j=y$, $c=\Sigma_{j=1}^{C}a_jb_j$가 되도록 설정한다. 각 참여자는 검증하려는 곱셈쌍의 share와 사전 곱셈쌍의 share를 가지고 다음과 같은 프로토콜을 수행한다.

1. 증명자는 C개의 랜덤 챌린지 $\epsilon_1,...,\epsilon_C\in\mathbb{F}$를 받는다. 비대화형 증명 시스템에서는 Fiat-Shamir 변환을 이용하여 증명자가 직접 챌린지를 생성한다.
2. i번째 참여자는 $\alpha_j^{(i)}=\epsilon_j\cdot x_j^{(i)}+a_j^{(i)}$, $j=1,...,C$를 내부적으로 계산한다.
3. 각 참여자는 $\alpha_1,...,\alpha_C$의 share를 공개하여 값을 복

구한다.

4. i번째 참여자는 $v^{(i)}$를 다음과 같이 내부적으로 계산한다.

$$v^{(i)} = \sum_{j=1}^{C} \epsilon_j \cdot z_j^{(i)} - \sum_{j=1}^{C} \alpha_j \cdot b_j^{(i)} + c^{(i)}$$

5. 각 참여자는 v의 share를 공개하여 값을 복구한 후, $v = 0$인지 확인한다.

만약 증명자가 올바른 곱셈 쌍 $(x_j, y_j, z_j)_{j=1}^{C}$ 및 사전 곱셈 쌍 $((a_j, b_j)_{j=1}^{C}, c)$를 이용했다면 $v = 0$이 성립한다. 하지만 잘못된 곱셈 쌍을 이용하여 프로토콜을 수행한 경우, $v = 0$이 성립할 확률은 $1/|\mathbb{F}|$이다. 여러 사전 곱셈 쌍을 이용하여 매번 새로운 랜덤 챌린지로 위 프로토콜을 반복하면 증명자가 올바른 곱셈 쌍을 가지고 있지 않고도 프로토콜을 통과할 확률을 원하는 만큼 낮출 수 있다.

2.2.2. Fiat-Shamir 변환

대화형 증명 시스템에서는 검증자가 랜덤 챌린지를 생성하여 증명자에게 전달하지만, 비대화형 증명 시스템을 만들기 위해서는 증명자가 직접 랜덤 챌린지를 생성해야 한다. Fiat-Shamir 변환은 대화형 증명 시스템을 비대화형 증명 시스템으로 변환하는 방법으로[11], 대화형 증명 시스템에서 검증자가 랜덤하게 선택하는 챌린지를 랜덤 오라클 접근으로 대체한다. 실제 구현 상에서는 현재 시점까지 얻어진 내부 정보를 모두 모은 후 해시값을 계산해 챌린지를 생성한다. 이렇게 하면 증명자는 스스로 랜덤 챌린지를 생성하지만 그 값을 자신이 원하는 값으로 마음대로 선택할 수는 없다.

2.2.3. 서명 크기 최적화

BN++ 프로토콜을 이용한 전자 서명에서 서명은 사용한 시드(seed), 곱셈 검증 프로토콜에서 사용하는 사전 곱셈 쌍 및 랜덤 챌린지, 커밋값 등으로 이루어진다. 따라서 서명 크기는 기본적으로 프로토콜의 반복 횟수, 검증하려는 회로의 곱셈 개수에 거의 비례하여 커진다.

BN++ 프로토콜에서는 서명 크기를 줄이기 위한 기법으로 반복 곱셈자(repeated multiplier) 기법을 제안한다. 이는 검증하려는 곱셈 쌍 중 동일한 값이 있을 경우, 공통의 사전 곱셈쌍을 이용하여 서명의 크기를 줄이는 방법이다. 예를 들어 검증하려고 하는 곱셈쌍이 (x_1, y, z_1) 및 (x_2, y, z_2)로 공통의 y를 사용한다면 1개의 사전 곱셈 쌍 (a, b, c)만 이용하여

$$\begin{cases} \alpha^{(i)} = \epsilon_1 \cdot x_1^{(i)} + \epsilon_2 \cdot x_2^{(i)} + a^{(i)} \\ v^{(i)} = \epsilon_1 \cdot z_1 + \epsilon_2 \cdot z_2 - \alpha \cdot y + c \end{cases}$$

로 정의한 후 $v = \Sigma_{i=1}^{N} v^{(i)} = 0$이 되는지 검증한다.

Ⅲ. AIMer 전자서명

AIMer 전자서명은 MPCitH 기반 영지식 증명 시스템인 BN++과 대칭키 프리미티브 기반 일방향 함수인 AIM이 결합되어 1.0 버전으로 제안되었다. 이후 일방향 함수 AIM에 대한 여러 제3자 분석이 제안됨에 따라 효율성을 유지하면서 안전성을 높일 수 있도록 구조를 변형한 AIM2가 제안되었고, 서명 크기 및 검증 시간을 줄일 수 있는 최적화 기법이 BN++ 시스템에 적용되어 2.0 버전으로 업데이트 되었다.

본 장에서는 AIMer 전자서명의 전반적인 작동 원리와, AIMer 2.0 버전에 적용된 BN++ 증명 시스템 최적화 기법 및 내부 일방향 함수 AIM2에 대해 소개한다.

3.1. 일방향 함수 AIM2

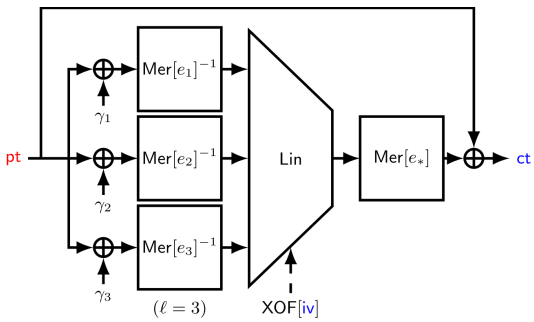
AIMer는 영지식 증명을 통해 검증할 일방향 함수로 IV 기반의 일방향 함수 $\mathrm{AIM2}$를 사용한다. $\mathrm{AIM2} : \{0,1\}^n \times \mathbb{F}_{2^n} \to \mathbb{F}_{2^n}$는 $\ell + 1$개의 지수 파라미터 $e_1, \ldots, e_\ell, e_* \in \mathbb{Z}$를 가지며, 다음과 같이 정의된다.

$$\mathrm{AIM2}(iv, pt) = \mathrm{Mer}[e_*] \circ \mathrm{Lin}[iv] \circ \mathrm{Mer}[e_1, \ldots, e_\ell]^{-1} \circ \mathrm{AddConst}(pt) \oplus pt.$$

각 내부 함수의 정의는 아래에 서술되어 있으며, 그림 2는 $\ell = 3$일 때의 AIM2 함수의 구조를 보여준

〔표 1〕 AIM2 파라미터

	λ	n	ℓ	e_1	e_2	e_3	e_*
AIM2-I	128	128	2	49	91	-	3
AIM2-III	192	192	2	17	47	-	5
AIM2-V	256	256	3	11	141	7	3



〔그림 2〕 $\ell=3$일 때의 AIM2 일방향 함수 구조.

다. 비도 $\lambda\in\{128,192,256\}$에 따른 파라미터는 표 1에 주어져있다.

3.1.1. AIM2 내부 함수 정의

먼저 AIM2의 비선형 연산은 Mersenne S-box $\mathrm{Mer}[e]$와 그 역함수 $\mathrm{Mer}[e]^{-1}$로 이루어져있다. $\mathrm{Mer}[e]:\mathbb{F}_{2^n}\to\mathbb{F}_{2^n}$는 유한체 상에서의 지수를 2^e-1로 가지는 거듭제곱 함수, 즉 $\mathrm{Mer}[e](x)=x^{2^e-1}$로 정의된다. 일대일 함수가 되도록 e를 선택하면 역함수 $\mathrm{Mer}[e]^{-1}$ 또한 거듭제곱 함수로 정의할 수 있으며, 지수 $\bar{e}$는 $\bar{e}=(2^e-1)^{-1}\pmod{2^n-1}$로 정의된다. 지수 e는 S-box가 일대일 함수가 되는 조건 뿐만 아니라 $\mathrm{Mer}[e]$가 $3n$개의 2차 방정식을 가지도록 선택한다. AIM2에서는 inverse Mersenne S-box를 직렬이 아닌 병렬로 적용하는데, 이를 나타내는 함수 $\mathrm{Mer}[e_1,\ldots,e_\ell]^{-1}:\mathbb{F}_{2^n}^\ell\to\mathbb{F}_{2^n}^\ell$는 다음과 같이 정의된다.

$$\mathrm{Mer}[e_1,\ldots,e_\ell]^{-1}(x_1,\ldots,x_\ell)=(\mathrm{Mer}[e_j]^{-1}(x_j))_{j=1}^{\ell}.$$

AIM2의 선형 연산은 상수 덧셈, 아핀 변환, 선형 되먹임으로 이루어져있다. 먼저 AIM2의 첫 번째 연산인 상수 덧셈 $\mathrm{AddConst}:\mathbb{F}_{2^n}\to\mathbb{F}_{2^n}^\ell$은 고정된 상수 $\gamma_1,\ldots,\gamma_\ell$에 대해 다음과 같이 정의된다[1].

$$\mathrm{AddConst}(x)=(x+\gamma_1,\ldots,x+\gamma_\ell).$$

상수 덧셈을 통해 이후 S-box에 들어가는 입력이 달라지게 만들어준다.

아핀 변환은 ℓn 비트의 내부 상태를 n 비트로 모아주는 역할을 한다. $\mathrm{Lin}[iv]:\mathbb{F}_{2^n}^\ell\to\mathbb{F}_{2^n}$은 입력으로 받은 초기화 벡터 iv를 통해 ℓ개의 $n\times n$ 랜덤 가역 행렬 $(A_{iv,j})_{j=1}^{\ell}$ 및 n 비트 랜덤 상수 b_{iv}를 생성한 후 다음과 같은 선형 연산을 수행한다.

$$\mathrm{Lin}[iv](x_1,\ldots,x_n)=\sum_{j=1}^{\ell}A_{iv,j}x_j+b_{iv}.$$

여기서 $A_{iv,j}x_j$는 x_j를 n 비트 벡터로 생각한 후 $A_{iv,j}$와 행렬-벡터곱 연산을 수행하는 것이다.

마지막으로 선형 되먹임은 $\mathrm{Mer}[e_*]$의 출력에 AIM2의 입력 pt를 더해준다. 선형 되먹임 연산을 통해 AIM2 연산은 가역성을 잃어버리며, 또한 입력값 pt가 모든 S-box의 입출력에 관여하도록 만들어준다.

3.1.2. AIM2 설계 사상

AIM2는 이진유한체 $\mathbb{F}_{2^n}$ 위에서 정의되어 있어 비트 단위의 연산을 사용하는 하드웨어 아키텍쳐에서 효율적으로 계산할 수 있게 설계되어 있다. 크기가 큰 유한체를 사용함으로써 BN++ 영지식 증명 시스템을 사용할 때 필요한 반복 회수를 줄일 수 있다.

또한 $\mathbb{F}_{2^n}$ 상에서는 입력을 비트 벡터로 보았을 때 행렬 곱셈 연산이 선형 연산이지만 다항식으로 표현했을 때 높은 차수를 가지므로, AIM2의 행렬 곱을 사용하는 아핀 변환은 서명의 크기를 증가시키지는 않지만 AIM2의 $\mathbb{F}_{2^n}$ 상에서의 차수는 크게 높일 수 있다. 아핀 변환에 사용되는 행렬과 상수 벡터는 iv값에 따라 랜덤하게 생성하므로 각 사용자마다 다른 아핀 변환을 사용하게 되며, 이는 multi-target 공격에 대한 저항성을 제공한다.

1) 상수 $\gamma_1,\ldots,\gamma_\ell$의 값은 KpqC 2라운드 제출 문서 혹은 [9]에서 확인할 수 있다.

AIM2의 S-box는 $y = x^{2^e-1}$ 형태의 Mersenne S-box 및 그 역함수를 사용한다. Mersenne S-box의 입출력을 검증하기 위해서는 $xy = x^{2^e}$가 성립하는지를 확인하면 되는데, $\mathbb{F}_{2^n}$ 상에서 $x \mapsto x^{2^e}$는 선형 연산이기 때문에 x의 share를 가진 각 참여자는 x^{2^e}의 share를 내부적으로 계산할 수 있다. Inverse Mersenne S-box의 경우는 비슷하게 $xy = y^{2^e}$가 성립하는지를 검증하면 된다. 자주 사용되는 대칭키 프리미티브인 inverse S-box $x \mapsto x^{2^n-2}$와 비교했을 때 Mersenne S-box가 가지는 장점은 각 S-box를 표현하는 2차 부울 방정식이 Mersenne S-box가 $3n$개로 $5n$개를 가지는 inverse S-box에 비해 적다는 것이다[2]. 입출력 쌍이 한 쌍 밖에 주어지지 않는 전자서명 환경에서는 대수적 공격이 가장 중요한 공격으로 작용하기 때문에 더 적은 개수의 방정식을 가지는 Mersenne S-box가 같은 서명 크기를 가지면서도 대수적 공격에 대해 더 안전할 것이라 기대할 수 있다. Mersenne S-box와 inverse Mersenne S-box는 서명 크기 상에서는 차이가 없지만, 병렬로 배치한 S-box를 inverse Mersenne S-box로 배치할 경우 AIM2를 묘사하는 대수적 시스템을 구성할 때 방정식의 개수가 더 적어진다[9].

서명 크기를 최소화 하기 위한 AIM2의 가장 중요한 특징은 입력값 pt가 내부 모든 S-box의 입출력에 사용되므로 BN++의 반복 곱셈자 기법을 모든 S-box 검증에 사용할 수 있다는 점이다. $\mathrm{Mer}[e_1,\ldots,e_\ell]^{-1}$에서는 pt에 서로 다른 상수가 더해져 입력으로 들어가고, $\mathrm{Mer}[\mathrm{e}_*]$에서는 되먹임 구조에 의해 pt에 ct를 더한 값이 출력으로 사용된다. pt에 더해지는 상수값들은 모두 공개된 상수값이므로 선형 연산으로 처리할 수 있다. 따라서 한 개의 사전 곱셈 쌍만 이용하여 AIM2 내의 모든 S-box 연산을 검증할 수 있다.

3.2. AIMer 전자서명 알고리즘

AIMer의 알고리즘은 키 생성, 서명, 검증으로 이루어져 있으며, 본 장에서는 각 알고리즘의 전반적인 구조를 설명한다. 정확한 알고리즘은 AIMer 홈페이지[3] KpqC 2라운드 제출 문서에서 확인할 수 있다. NIST에서 제시하는 전자서명 비도 L1, L3, L5에 따른 추천 파라미터는 표 2에 정리되어 있다.

〔표 2〕 AIMer 버전 2.0의 파라미터. 비도 L1, L3, L5는 각각 128, 192, 256 비트의 비도를 나타내며, N과 τ는 각각 참여자 수와 반복 수행 횟수를 의미한다.

비도	파라미터	일방향 함수	해시	N	τ
L1	aimer128f	AIM2-I	SHAKE128	16	33
	aimer128s	AIM2-I	SHAKE128	256	17
L3	aimer192f	AIM2-III	SHAKE256	16	49
	aimer192s	AIM2-III	SHAKE256	256	25
L5	aimer256f	AIM2-V	SHAKE256	16	65
	aimer256s	AIM2-V	SHAKE256	256	33

3.2.1. 키 생성

λ 비트 비도에 대해 키 생성 단계는 다음과 같다.

1. 랜덤한 λ 비트값 pt와 iv를 생성한다.
2. $ct = \mathrm{AIM2}(iv, pt)$를 계산한다.
3. 비밀키 (pt, iv, ct), 공개키 (iv, ct)를 출력한다.

3.2.2. 서명 알고리즘

서명 알고리즘은 MPCitH 환경에서 AIM2를 검증하고, 한 명을 제외한 나머지 참여자의 내부 정보를 공개하여 서명으로 출력한다. 다음의 총 5단계(phase)로 이루어져 있다.

단계 1. 각 참여자의 시드 생성 및 내부 관점 커밋
메시지의 해시값, AIM2의 입력값 pt,서명의 랜덤값을 기반으로 루트 시드와 솔트(salt)를 생성한다. 이후 MPC 프로토콜의 각 반복 수행에 대해 각 참여자의 시드와 랜덤 테이프(tape)를 생성 및 커밋하고, AIM2 내부 중간값 및 사전 곱셈 쌍의 share를 생성한다.

단계 2. 곱셈 검증 프로토콜을 위한 챌린지 생성
첫 번째 챌린지 해시값을 계산한 후 이를 확장하여 각 반복 수행에서 곱셈 검증 과정에서 사용할 챌린지들을 생성한다.

2) 정확히는 $3n$개의 2차 부울 방정식을 가지는 e를 선택하여 사용한다.

3) https://aimer-signature.org

단계 3. 곱셈 검증 프로토콜 과정 커밋
곱셈 검증 프로토콜 과정을 수행하여 각 참여자 i의 $\alpha^{(i)}$ 및 $v^{(i)}$ 값들을 계산한다.

단계 4. 비공개할 참여자를 선택하는 챌린지 생성
두 번째 챌린지 해시값을 계산하고 이를 확장하여 각 반복 수행에서 공개하지 않을 참여자 $\bar{i}$를 결정한다.

단계 5. 나머지 참여자의 내부 관점 공개
전체 솔트값 및 두 개의 챌린지 해시값, 각 반복 수행에서 공개할 참여자들의 시드를 공개하여 내부 관점을 다시 계산할 수 있게 해주고, 비공개되는 참여자 $\bar{i}$의 경우 단계 1에서 생성한 커밋값과 단계 3에서 계산한 $\alpha^{(i)}$ 값만 서명에 포함된다.

3.2.3. 검증 알고리즘

검증 알고리즘은 서명에 기록되어 있는 MPCitH 기반의 검증 과정이 제대로 이루어졌는지 확인하는 과정이다.

1. 서명 단계 1, 2 재수행
   각 반복 수행에서 공개된 참여자들의 시드를 이용해 서명 단계 1에서 생성하는 값들을 다시 계산한다. 이후 이를 바탕으로 서명 과정을 재수행하여 첫 번째 챌린지 해시값을 계산한다.
2. 서명 단계 3, 4 재수행
   각 반복 수행에서 곱셈 검증 과정을 수행하여 공개되지 않은 참여자 $\bar{i}$의 $v^{(\bar{i})}$값을 $v=0$을 이용하여 복구하고, 이를 통해 두 번째 챌린지 해시값을 계산한다.
3. 복구한 두 개의 챌린지 해시값이 서명에 있는 값과 일치하는지 검증한다.

3.3. 개선된 BN++ 증명 시스템

AIMer 버전 1.0에서는 기존의 BN++ 증명 시스템을 그대로 이용했지만, 버전 2.0에서는 그 동안 여러 최적화 기법을 적용하여 개선된 BN++ 증명 시스템을 사용하게 되었다. 이에 따라 AIM에서 AIM2로 변경하는 과정에서 파라미터 변화 등으로 인한 성능 저하가 예상되었음에도 불구하고 서명 및 검증 시간을 줄일 뿐 아니라 서명의 크기도 줄였다.

BN++ 프로토콜에서 시간을 많이 차지하는 부분은 해시 연산인데, 이에 관련한 연산에 많은 최적화 기법이 적용되었다. 가상 MPC 환경의 각 참여자의 시드를 생성하는 과정에서 GGM 트리를 이용한 최적화 기법을 적용하였고[12], 서명의 메시지 부분에 대한 해시 연산을 먼저 수행하도록 변경하였으며, Commit 및 ExpandTape 과정을 하나로 합쳐서 불필요한 공회전을 줄였다.

서명 크기 측면에서는 multi-target 공격에 대한 방어를 위해 λ 비트 비도를 위해서 항상 2λ 비트의 salt를 추가하고 있었는데, salt는 항상 iv와 같이 사용됨을 이용해 λ 비트의 salt만 사용하고도 원하는 안전성을 얻을 수 있게 하였다.

Ⅳ. 성능 비교

AIMer 전자서명의 성능을 측정하기 위해 Intel Xeon E5-1650 v3 @ 3.50 GHz의 CPU 환경에서 AVX2 최적화를 사용하였다. 표 3에는 AIMer 버전 2.0의 파라미터별 성능이 정리되어 있다.

128비트 비도를 가지는 파라미터에 대해 AIMer와 다른 전자서명의 성능을 비교한 결과는 표 4에 정리되어 있다. 현재 NIST 표준으로 선정된 전자서명 및 KpqC 2라운드 전자서명 후보 알고리즘에 대해 성능을 비교하였으며, 기반문제 별로 성능을 정리하였다. 공개키와 서명을 합한 크기는 격자 문제에 기반하지 않은 전자서명 중에서는 AIMer가 가장 작은 크기를 가지는 것을 확인할 수 있으며, 특히 동일하게 대칭키 기반 전자서명이라고 할 수 있는 해시 기반 전자서명 SPHINCS+와 비교했을 때는 aimer128f가 서명 크기 뿐만 아니라 서명 및 검증 시간도 더 빠른 것을 확인

〔표 3〕 AIMer 버전 2.0의 파라미터별 성능. |sk|, |pk|, |sig|는 각각 비밀키, 공개키, 서명의 크기를 의미한다.

	키 생성 (ms)	서명 (ms)	검증 (ms)	|sk| (B)	|pk| (B)	|sig| (B)
aimer128f	0.03	0.42	0.41	48	32	5888
aimer128s	0.03	3.18	3.13	48	32	4160
aimer192f	0.05	1.03	1.03	72	48	13056
aimer192s	0.05	7.94	7.86	72	48	9120
aimer256f	0.10	2.07	2.03	96	64	25120
aimer256s	0.10	15.26	14.81	96	64	17056

〔표 4〕 전자서명 별 성능 비교. 서명 및 검증 시간은 평균값을 사용하였다. †KpqC 2라운드 후보 알고리즘의 경우 cpu cycle 단위의 성능을 동일한 환경에서 ms 단위로 환산한 결과이다.

기반 문제	서명	공개키 크기 (B)	서명 크기 (B)	서명 시간 (ms)	검증 시간 (ms)
격자 문제	Dilithium2	1312	2420	0.10	0.03
	Falcon-512	897	690	0.27	0.04
	HAETAE-120†	992	1474	0.56	0.03
	NCC-Sign-cyclo† (ref)	1564	2458	0.24	0.06
다변수 방정식	MQ-Sign-RR†	328441	134	0.05	0.02
해시	SPHINCS+-128s	32	7856	315.74	0.35
	SPHINCS+-128f	32	17088	16.32	0.97
대칭키	aimer128s	32	4160	3.18	3.13
	aimer128f	32	5888	0.42	0.41

할 수 있다.

Ⅴ. 결 론

AIMer는 격자 문제에 기반하지 않은 양자 내성 전자서명으로 MPCitH 기반 영지식 증명과 대칭키 프리미티브 기반 일방향 함수를 결합하여 만들어진 전자서명이다. 크기가 큰 유한체 상에서 효율적으로 작동하는 MPCitH 기반 영지식 증명 시스템인 BN++을 개선하였고, 해당 증명 시스템에서 효율적으로 검증할 수 있는 대칭키 프리미티브 기반 일방향 함수 AIM2를 설계하여 서명 크기를 최소화 하였다. 이러한 최적화 기법을 통해 AIMer는 공개키와 서명을 합쳤을 때 NIST 표준 선정 및 KpqC 2라운드 후보 알고리즘 중 비격자 문제 기반 카테코리에서 가장 작은 크기를 가진다.

초기 버전에서는 큰 메모리 사용량이 경량 환경에서의 한계점으로 지적되기도 하였으나[13] 현재는 메모리 최적 구현을 통해 전반적인 메모리 사용량이 크게 줄였으며, ARM Cortex-M4 환경에서도 모든 파라미터가 구현될 수 있음을 확인하였다. 추후 이러한 다양한 환경에서의 최적화 연구 또한 진행될 수 있을 것으로 기대된다.

참 고 문 헌

[1] Vadim Lyubashevsky, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Peter Schwabe, Gregor Seiler, Damien Stehlé, and Shi Bai. “CRYSTALS-DILITHIUM”. Technical report, National Institute of Standards and Technology, 2022.

[2] Thomas Prest, Pierre-Alain Fouque, Jeffrey Hoffstein, Paul Kirchner, Vadim Lyubashevsky, Thomas Pornin, Thomas Ricosset, Gregor Seiler, William Whyte, and Zhenfei Zhang. “FALCON”. Technical report, National Institute of Standards and Technology, 2022.

[3] Andreas Hulsing, Daniel J. Bernstein, Christoph Dobraunig, Maria Eichlseder, Scott Fluhrer, Stefan-Lukas Gazdag, Panos Kampanakis, Stefan Kolbl, Tanja Lange, Martin M. Lauridsen, Florian Mendel, Ruben Niederhagen, Christian Rechberger, Joost Rijneveld, Peter Schwabe, Jean-Philippe Aumasson, Bas Westerbaan, and Ward Beullens. “SPHINCS+”. Technical report, National Institute of Standards and Technology, 2022.

[4] Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai. “Zero-knowledge from Secure Multiparty Computation”. *ACM STOC* 2007, pages 21 – 30, 2007.

[5] Daniel Kales and Greg Zaverucha. “Efficient Lifting for Shorter Zero-Knowledge Proofs and Post-Quantum Signatures”. *Cryptology ePrint Archive*, Paper 2022/588, 2022.

[6] Seongkwang Kim, Jincheol Ha, Mincheol Son,

Byeonghak Lee, Dukjae Moon, Joohee Lee, Sangyub Lee, Jihoon Kwon, Jihoon Cho, Hyojin Yoon, and Jooyoung Lee. “AIM: Symmetric Primitive for Shorter Signatures with Stronger Security”. *In Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, page 401 – 415. Association for Computing Machinery, 2023

[7] Fukang Liu, Mohammad Mahzoun, Morten Øygarden, and Willi Meier. “Algebraic Attacks on RAIN and AIM Using Equivalent Representations”. *IACR Transactions on Symmetric Cryptology*, 2023(4):166 – 186, Dec. 2023.

[8] Kaiyi Zhang, Qingju Wang, Yu Yu, Chun Guo, and Hongrui Cui. “Algebraic Attacks on Round-Reduced Rain and Full AIM-III”. In *ASIACRYPT 2023*, pages 285 – 310. Springer, 2023.

[9] Seongkwang Kim, Jincheol Ha, Mincheol Son, and Byeonghak Lee. “Efficacy and Mitigation of the Cryptanalysis on AIM”. *Cryptology ePrint Archive*, Paper 2023/1474, 2024.

[10] Carsten Baum and Ariel Nof. “Concretely Efficient Zero-Knowledge Arguments for Arithmetic Circuits and Their Application to Lattice Based Cryptography”. In *PKC 2020*, pages 495 – 526. Springer, 2020.

[11] Amos Fiat and Adi Shamir. “How to prove yourself: Practical solutions to identification and signature problems”. In *CRYPTO’ 86*, pages 186 – 194. Springer, 1987.

[12] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. “How to construct random functions”. *Journal of the ACM (JACM)*, 33(4):792 – 807, 1986.

[13] Matthias J. Kannwischer, Markus Krausz, Richard Petri, and Shang-Yi Yang. “pqm4: Benchmarking nist additional post-quantum signature schemes on microcontrollers”. *Cryptology ePrint Archive*, Paper 2024/112, 2024.

〈저 자 소 개〉

하 진 철 (Jincheol Ha)

2019년 2월 : 한국과학기술원 수리과학과 졸업
2021년 2월 : 한국과학기술원 정보보호대학원 석사 졸업
2021년 3월~현재 : 한국과학기술원 정보보호대학원 박사과정
<관심분야> 암호학

김 성 광 (Seongkwang Kim)

2016년 2월 : 포항공과대학교 수학과 학부 졸업
2018년 2월 : 한국과학기술원 수리과학부 석사 졸업
2022년 2월 : 한국과학기술원 정보보호대학원 박사 졸업
2022년 3월~현재 : 삼성 SDS 재직 중
<관심분야> 암호학

손 민 철 (Mincheol Son)

2020년 2월 : 고려대학교 사이버국방학과 졸업
2022년 8월 : 한국과학기술원 정보보호대학원 석사 졸업
2022년 9월~현재 : 한국과학기술원 정보보호대학원 박사과정
<관심분야> 암호학