디지털 서명 알고리즘 AIMer

Version 3.0

저자 권지훈 (삼성에스디에스)

문덕재 (삼성에스디에스)

윤효진 (삼성에스디에스)

이병학 (삼성에스디에스)

이상엽 (삼성에스디에스)

조지훈 (삼성에스디에스)

손민철 (한국과학기술원)

이주영 (한국과학기술원)

김성광 (고려대학교)

이주희 (성신여자대학교)

하진철 (숭실대학교)

메일 team@aimer-signature.org

홈페이지 https://aimer-signature.org

목 차

디지털 서명 알고리즘 AIMer

Digital signature algorithm AIMer

1 적용범위

이 표준에서는 디지털 서명 알고리즘 AIMer[1]의 전체 구조와 동작 방식을 명시한다. AIMer는 매개변수에 따라 세부 알고리즘 AIMer-128f, AIMer-128s, AIMer-192f, AIMer-192s, AIMer-256f, AIMer-256s으로 구성된다. 또한, AIMer 알고리즘의 핵심 구성 요소인 일방향 함수 AIM3의 구조와 세부 논리를 명시한다.

2 인용표준

다음의 인용표준은 전체 또는 부분적으로 AIMer의 적용을 위해 필수적이다. 발행연도가 표기된 인용표준은 인용된 판만을 적용한다. 발행연도가 표기되지 않은 인용표준은 최신판(모든 추록을 포함)을 적용한다.

ISO/IEC 10118-3, Information technology — Security techniques — Hash-functions
KS X ISO/IEC 10118-1, 정보기술 — 보안기술 — 해시 함수
KS X ISO/IEC 11770-3, 정보기술 — 보안기술 — 키 관리 — 제3부: 비대칭 기법을 이용한 메커니즘
KS X ISO/IEC 11770-5, 정보기술 — 보안기술 — 키 관리 — 제5부: 그룹 키 관리
KS X ISO/IEC 14888-1, 정보기술 — 보안기술 — 부가형 디지털 서명 — 제1부: 일반
KS X ISO/IEC 18033-2, 정보기술 — 보안기술 — 암호 알고리즘 — 제2부: 비대칭형 암호

3 용어와 정의

이 표준의 목적을 위하여 다음의 용어와 정의를 적용한다.

3.1
데이터 요소(data element)
정수, 비트열, 정수의 집합 또는 비트열의 집합

[출처: KS X ISO/IEC 14888-1:2008, **3.3**, 수정]

3.2
확장 출력 함수(extendable-output function)
XOF
다음의 두 가지 성질을 만족하는 임의의 길이의 비트열을 임의의 길이의 비트열로 변환하는 함수

- 주어진 출력에 대하여, 이러한 출력을 만드는 입력을 찾기가 계산적으로 불가능함
- 주어진 입력에 대하여, 동일한 출력을 만드는 제 2의 입력을 찾기가 계산적으로 불가능함

비고 계산적 불가능성은 보안 요구사항 및 환경에 따라 달라질 수 있다.

3.3
해시 값(hash-value)
해시 함수의 출력인 비트열

비고 이 분야의 문헌에서는 해시 코드와 동일하거나 유사한 의미로 사용되는 다양한 용어가 있다. 예를 들어, 수정 탐지 코드(modification detection code), 조작 탐지 코드(manipulation detection code), 다이제스트(digest), 해시 결과(hash-result), 해시 코드(hash-code), 임프린트(imprint)가 있다.

[출처: KS X ISO/IEC 10118-1:2000, **3.4**, 수정]

3.4
해시 함수(hash-function)
다음의 두 가지 성질을 만족하는 임의의 길이의 비트열을 고정된 길이의 비트열로 변환하는 함수

- 주어진 출력에 대하여, 이러한 출력을 만드는 입력을 찾기가 계산적으로 불가능함
- 주어진 입력에 대하여, 동일한 출력을 만드는 제 2의 입력을 찾기가 계산적으로 불가능함

비고 계산적 불가능성은 보안 요구사항 및 환경에 따라 달라질 수 있다.

[출처: ISO/IEC 10118-1:2000, **3.5**, 수정]

3.5
메시지(message)
임의 길이의 비트열

[출처: KS X ISO/IEC 14888-1:2008, **3.10**]

3.6
매개변수(parameter)
알고리즘의 상세 작동 과정을 결정하는 정수, 비트열이나 해시 함수

[출처: KS X ISO/IEC 14888-1:2008, **3.11**, 수정]

3.7
서명(signature)
서명 과정에서 발생하는 하나 이상의 데이터 요소

[출처: KS X ISO/IEC 14888-1:2008, **3.12**, 수정]

3.8
서명 키(signature key)
서명 과정에서 어떤 개체만이 이용할 수 있는 고유한 비밀 데이터 요소의 집합

[출처: KS X ISO/IEC 14888-1:2008, **3.13**, 수정]

3.9
서명 과정(signature process)
입력으로 메시지, 서명 키 그리고 도메인 매개변수를 취하고 출력으로 서명을 생성하는 과정

[출처: KS X ISO/IEC 14888-1:2008, **3.14**]

3.10
검증 키(verification key)
어떤 개체의 서명 키와 수학적으로 연관이 되며 검증 과정에서 사용되는 공개 데이터 요소의 집합

[출처: KS X ISO/IEC 14888-1:2008, **3.16**, 수정]

3.11
검증 과정(verification process)
입력으로 서명, 메시지, 검증 키 그리고 도메인 매개변수를 취하여 출력으로 서명 검증 결과로써 유효 또는 무효 값을 생성하는 과정

[출처: KS X ISO/IEC 14888-1:2008, **3.17**, 수정]

3.12
키 쌍(key pair)
서명 키와 검증 키 쌍

[출처: KS X ISO/IEC 14888-1:2008, **3.9**, 수정]

3.13
일방향 함수(one-way function)
주어진 입력에 대한 출력을 계산하는 것은 용이하지만, 주어진 출력에 대응하는 입력을 찾는 것은 계산적으로 불가능한 성질을 가지는 함수

[출처: KS X ISO/IEC 11770-3:2015, **3.30**, 수정]

3.14
노드
트리의 요소

[출처: ISO/IEC 23090-9:2023, **3.3.7**]

3.15
형제 노드
같은 부모 노드를 가지는 노드

[출처: ISO/IEC 23090-9:2023, **3.3.13**, 수정]

3.16
(노드 w의) 자식 노드(child node)
노드 w에 인접하여 있고, 루트 노드와의 단일 경로에 연결되어 있는 노드

[출처: KS X ISO/IEC 11770-5:2014, **3.7**, 수정]

3.17
(노드 c의) 부모 노드
노드 c의 상위에 연결된 노드

[출처: KS X ISO/IEC 11770-5:2014, **3.22**]

3.18
단말 노드
자식 노드가 없는 노드

[출처: KS X ISO/IEC 11770-5:2014, **3.17**, 수정]

3.19
루트 노드
다른 노드의 자식 노드가 아닌 노드

[출처: KS X ISO/IEC 11770-5:2014, **3.27**, 수정]

3.20
트리
루트 노드라 부르는 특정 노드를 가지는 연결된 비순환 그래프

[출처: KS X ISO/IEC 11770-5:2014, **3.30**, 수정]

3.21
***d*-진(ary) 트리**
단말 노드를 제외한 각 노드에 d개 자식 노드가 있는 트리

보기 각 노드에 2개의 자식 노드가 있는 트리를 이진 트리라고 부른다.

[출처: KS X ISO/IEC 11770-5:2014, **3.8**, 수정]

4 기호 및 표기

4.1 수학적 표기

이 표준에서는 다음과 같은 수학적 표기들이 사용된다.

$[n]$	자연수 n에 대하여, 집합 $\{0,1,\dots,n-1\}$을 의미함
$\{0,1\}^n$	자연수 n에 대하여, n비트 길이를 가지는 모든 비트열의 집합
$\{0,1\}^*$	임의의 길이를 가지는 모든 비트열의 집합
0^n	모든 자리수가 0인 길이 n의 비트열
$a\|\|b$	벡터 혹은 비트열 a와 b를 그 순서대로 연접
$a \leftarrow b$	b의 값을 a 변수에 대입
$a \leftarrow_{\$} S$	a를 집합 S에서 무작위로 선택
$a \oplus b$	같은 길이를 가지는 비트열 a와 b의 배타적 논리합(XOR)
$a \gg i$	비트열 a를 최상위 비트를 가장 왼쪽으로 봤을 경우 i만큼 오른쪽 비트 시프트한 (즉, j번 째 비트를 $(j-i)$번 째 위치로 이동) a와 같은 길이의 비트열(최상위 비트 i개는 0으로 채움)
$a \ll i$	비트열 a를 최상위 비트를 가장 왼쪽으로 봤을 경우 i만큼 왼쪽 비트 시프트한 (즉, j번 째 비트를 $(j+i)$번 째 위치로 이동) a와 같은 길이의 비트열(최하위 비트 i개는 0으로 채움)
$\log m$	정수 m에 대해 밑이 2인 로그 값
$v[a]$	벡터 v의 a번째 원소(a의 인덱스는 0부터 사용)

$v[a:b]$	벡터 v의 a번째(포함)부터 b번째(불포함)까지의 슬라이스
$x[a]_{\text{bit}}$	비트열 x의 a번째 비트
$x[a:b]_{\text{bit}}$	비트열 x의 a번째(포함)부터 b번째(불포함)까지의 슬라이스. 이진 행렬의 경우 별도의 표기를 활용함(**5.3** 참조)
$\lvert x \rvert$	비트열 x의 길이
$\lfloor x \rfloor$	실수 x에 대하여 x보다 크지 않은 가장 큰 정수
$(a_i)_{i\in[n]}$	자연수 n, 수열 a_i에 대하여 배열 $(a_0, a_1, \ldots, a_{n-2}, a_{n-1})$을 의미함
SHAKE_λ	보안 매개변수 λ에 따라 정의된 확장 출력 함수(**8.1** 참조)

4.2 기호

이 표준에서는 다음과 같은 기호들이 사용된다. 일방향 함수 AIM3 및 S-box의 세부 논리는 **6.1**에 기술되어 있다.

λ	보안 매개변수(security parameter)
n	AIM3 S-box의 입출력 비트 길이(항상 보안 매개변수 λ와 동일)
ℓ	AIM3 첫 번째 라운드의 S-box 개수
τ	영지식증명에서의 반복(repetition) 수
N	영지식증명에서의 가상참가자(party) 수
$\perp$	알고리즘이 실패하거나 유효한 출력을 생성하지 못했음을 나타내는 기호

5 수학적 규정

5.1 비트와 비트열

비트는 기호 0 또는 1 중에 하나이며, 비트열은 비트들의 수열이다. $b_0, b_1, \ldots, b_{n-1}$이 비트라면, $b = b_0 b_1 \ldots b_{n-1}$는 주어진 순서대로 구성된 길이 n의 비트열이다.

5.1.1 비트열과 정수 변환

비트열 $b = b_0 b_1 \ldots b_{n-1}$와 정수 x의 대응은 다음과 같이 정의한다.

$$x = b_0 \cdot 2^0 + b_1 \cdot 2^1 + \cdots + b_{n-1} \cdot 2^{n-1}.$$

여기에서 비트 b_0은 최하위 비트(least significant bit), 비트 b_{n-1}은 최상위 비트(most significant bit)에 해당한다.

정수를 비트열로 변환하는 함수 IntToBits(알고리즘 1)는 음이 아닌 정수 x와 출력될 비트 길이 s를 입력으로 받아 s비트 길이의 비트열 b를 출력한다.

알고리즘 1 $\text{IntToBits}(x, s)$ — 정수를 비트열로 변환

입력: 음이 아닌 정수 x, 양수 s
출력: s비트 길이의 비트열 b

1: $z \leftarrow x$
2: **for** i **from** 0 **to** $s - 1$ **do**
3: $\quad b[i] \leftarrow z \bmod 2$

4: $z \leftarrow \lfloor z/2 \rfloor$

5: **end for**

6: **return** b

비트열을 정수로 변환하는 함수 BitsToInt(알고리즘 2)는 s비트 길이의 비트열 b를 입력으로 받아 음이 아닌 정수 x를 출력한다.

알고리즘 2 $\text{BitsToInt}(b, s)$ — 비트열을 정수로 변환

입력: s비트 길이의 비트열 b
출력: 음이 아닌 정수 x

1: $x \leftarrow 0$

2: **for** i **from** 1 **to** s **do**

3: $x \leftarrow 2x + b[s - i]$

4: **end for**

5: **return** x

이 표준에서 정수는 10진수 또는 16진수 형태로 사용하며, 16진수 값은 앞에 `0x`를 붙여 표기한다. 예를 들어, 16진수 값 `0xa34b`은 IntToBits 함수를 사용하여 **표 1**과 같이 비트열 $b = b_0 b_1 \dots b_{15}$로 변환할 수 있다.

표 1 — 16진수 값과 비트열 변환 예제

`0xa`				`0x3`				`0x4`				`0xb`			
1	0	1	0	0	0	1	1	0	1	0	0	1	0	1	1
b_{15}	b_{14}	b_{13}	b_{12}	b_{11}	b_{10}	b_9	b_8	b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0

5.2 유한체

유한체(finite field)는 덧셈 및 곱셈 연산에 대하여 닫혀 있으며, 항등원과 역원이 존재하고, 분배법칙이 성립하는 체(field) 중 원소의 개수가 유한한 체를 말한다. 유한체의 원소 개수는 소수 p 또는 소수의 거듭제곱 p^n 형태를 가지며, 유한체는 $\text{GF}(p^n)$으로 표기한다.

이 표준에서 사용하는 유한체 $\text{GF}(2^n)$은 계수 0과 1을 가지는 다항식 집합을 차수 n의 기약다항식 $f(X)$에 대한 나머지 연산으로 정의한다.

이 표준에서 사용하는 유한체와 유한체 연산 알고리즘은 **8.2**에 기술되어 있다.

5.2.1 비트열과 유한체 원소 변환

n비트 길이의 비트열 $b = b_0 b_1 \dots b_{n-1}$를 유한체 $\text{GF}(2^n)$의 원소 $x(X)$로 변환하는 함수는 다음과 같이 정의한다.

$$x(X) = b_0 \cdot X^0 + b_1 \cdot X^1 + \cdots + b_{n-1} \cdot X^{n-1}.$$

비트열 b의 각 비트 $b_i (0 \le i < n)$는 다항식 $x(X)$에서 X^i 항의 계수에 대응한다. 즉, n비트 길이의 비

트열은 유한체 $\mathrm{GF}(2^n)$의 원소로 해석할 수 있다.

이 표준에서는 유한체 $\mathrm{GF}(2^n)$의 차수 n이 보안 매개변수 λ와 항상 동일하기 때문에, 비트열 $\{0,1\}^\lambda$는 유한체 $\mathrm{GF}(2^n)$의 원소인 비트열 $\{0,1\}^n$과 동일하게 간주한다.

5.2.2 유한체 연산

유한체 $\mathrm{GF}(2^n)$의 두 원소 x와 y의 덧셈은, 해당 원소를 이루는 비트열 간의 비트별 배타적 논리합으로 정의한다.

$$x + y = x \oplus y.$$

유한체 $\mathrm{GF}(2^n)$의 두 원소 x와 y의 곱셈은, 다항식 $x(X)$, $y(X)$로 대응하여 곱셈을 계산한 뒤, 기약다항식 $f(X)$로 나눈 나머지로 정의한다.

$$x \cdot y = x(X) \cdot y(X) \bmod f(X).$$

5.3 이진 행렬

이진 행렬(binary matrix)은 모든 성분이 0과 1로 이루어진 행렬을 말한다. 이진 행렬에서의 연산은 유한체 $\mathrm{GF}(2)$ 위에서 정의되며, 각 성분별 덧셈은 배타적 논리합, 곱셈은 논리곱으로 계산한다.

이 표준에서 이진 행렬은 이중 배열로 표기한다. 첫번째 인덱스(index)는 행 번호를, 두번째 인덱스는 열 번호를 가리킨다. 예를 들어, $m \times n$ 행렬 A에 대해, $A[i][j]$는 (i,j)번째 원소를 의미한다.

5.4 코딩 관습

알고리즘의 코드 형식에서 반복자 표기 for i from a to b는 a와 b를 모두 포함하여 반복한다. 예를 들어, for i from 1 to 4는 $i = 1, 2, 3, 4$ 순서로 반복한다.

각 함수의 명세에 기술된 알고리즘은, 동일한 입력에 대해 동일한 결과를 출력하는 다른 알고리즘으로 대체될 수 있다.

6 일방향 함수 AIM3

6.1 일반사항

일방향 함수 AIM3: $\{0,1\}^n \times \{0,1\}^\lambda \to \{0,1\}^n$는 n비트 길이의 비트열 pt와 λ비트 길이의 비트열 iv를 입력으로 받아 n비트 길이의 비트열 ct를 출력한다. 입력 pt는 서명 키의 일부이며, 서명자 이외에는 비밀로 유지되어야 한다. 입력 iv와 출력 ct는 검증 키로 사용되며, 서명 키의 일부를 구성한다.

일방향 함수 AIM3는 S-box의 입출력 길이 n, iv 입력 길이 λ, 첫 번째 라운드 S-box의 수 ℓ, S-box 계산에 입력되는 거듭제곱 지수를 결정하는 $(e_0, \ldots, e_\ell)$를 매개변수(**7.2** 참조)로 사용한다.

일방향 함수 AIM3의 S-box는 n비트 길이의 비트열 x를 입력으로 받아 n비트 길이의 비트열 y을 계산하며, 다음과 같이 표기한다.

$$y = S[e](x) = x^{\overline{e}} + x^{e_{inv}}.$$

여기에서 $\overline{e} = 2^e - 1$이고 $e_{inv} = 2^n - 2$이다.

AIM3의 S-box는 유한체 거듭제곱 함수 Exp(**8.2.2** 및 알고리즘 16 참조)를 사용하여 계산한다.

AIM3의 선형층은 임의의 이진 행렬과 이진 벡터로 구성되며 선형층의 생성 방법은 AIM3_GenerateLinear(**6.2.1** 및 알고리즘 4 참조) 함수를 참조한다.

일방향 함수 AIM3는 보안 매개변수 λ에 따라 AIM3-I($\lambda = 128$), AIM3-III($\lambda = 192$), AIM3-V($\lambda = 256$)로 구분하며, AIM3-I을 도식화한 그림은 **그림 1**에 제시되어 있다.

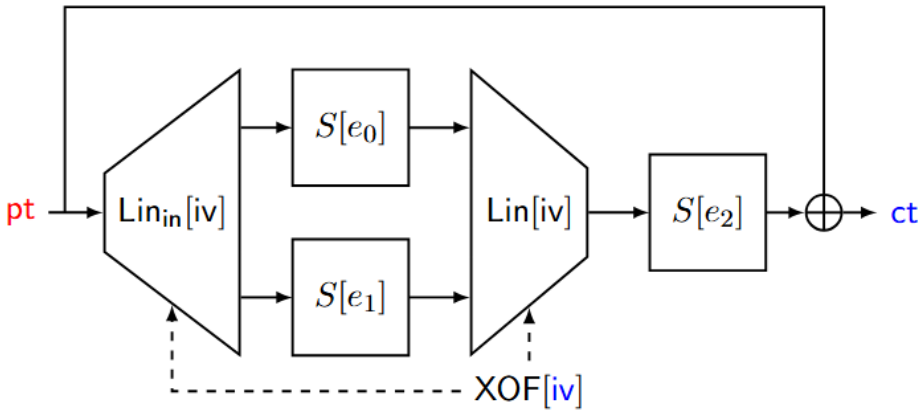


그림 1 — AIM3-I의 도식

일방향 함수 AIM3는 다음과 같이 계산한다.

- AIM3 선형층 생성 함수 AIM3_GenerateLinear(**6.2.1** 및 알고리즘 4 참조)에 iv를 입력하여 이진 행렬 $(A_j)_{j\in[2\ell]} \in (\{0,1\}^{n\times n})^{2\ell}$와 벡터 $(b_j)_{j\in[\ell+1]} \in (\{0,1\}^n)^{\ell+1}$를 생성한다.
- 입력 비트열 pt로부터 S-box 입력 값 $x_i = A_i \cdot pt + b_i$ $(0 \le i < \ell)$을 계산한다.
- 각 $x_i (0 \le i < \ell)$를 S-box에 입력하여 $y_i = S[e_i](x_i)$ $(0 \le i < \ell)$를 계산한다.
- 선형 함수 $\text{Lin}(y_0 \| \ldots \| y_{\ell-1}) = b_\ell + \sum_{j\in[\ell]} A_{j+\ell} \cdot y_j$를 수행하고 그 결과를 x_ℓ라고 한다.
- $y_\ell = S[e_\ell](x_\ell)$을 계산한 후, 입력 비트열 pt를 더하여 AIM3의 출력 ct를 계산한다.

6.2 AIM3 보조 함수

6.2.1 AIM3 0입력 S-box 제외 일방향 함수 AIM3_NoZeroInpSB

AIMer에서는 AIM3의 S-box에 0이 입력되지 않는 경우만 사용되기 때문에 0을 입력값으로 갖는 S-box가 있을 경우 비트열 ct 대신 실패문자 $\perp$을 출력하는 변형된 일방향 함수 AIM3_NoZeroInpSB(알고리즘 3)를 키 생성 과정에 사용된다.

알고리즘 3 AIM3_NoZeroInpSB(pt, iv) — S-box에 0이 입력될 경우 $\perp$을 출력하도록 변형된 AIM3

입력: 비트열 $pt \in \{0,1\}^n$, 비트열 $iv \in \{0,1\}^\lambda$
출력: 비트열 $ct \in \{0,1\}^n$ 혹은 $\perp$

1: $\left((A_j)_{j\in[2\ell]}, (b_j)_{j\in[\ell+1]}\right) \leftarrow$ AIM3_GenerateLinear(iv) ▷ **6.2** 및 알고리즘 4 참조
2: $x_\ell \leftarrow b_\ell$
3: **for** j **from** 0 **to** $\ell - 1$ **do**

4: $x_j \leftarrow \mathrm{MatVecMul}(A_j, pt) + b_j$ ▷ **8.2.3** 및 알고리즘 17 참조

5: $y_j \leftarrow \mathrm{Exp}(x_j, 2^{e_j} - 1) + \mathrm{Exp}(x_j, 2^n - 2)$ ▷ **8.2.2** 및 알고리즘 16 참조

6: $x_\ell \leftarrow x_\ell + \mathrm{MatVecMul}(A_{\ell+j}, y_j)$ ▷ **8.2.3** 및 알고리즘 17 참조

6: **end for**

7: $\mathrm{y}_\ell \leftarrow \mathrm{Exp}(x_\ell, 2^{e_\ell} - 1) + \mathrm{Exp}(x_\ell, 2^n - 2)$ ▷ **8.2.2** 및 알고리즘 16 참조

8: $ct \leftarrow pt + y_\ell$

9: **return** ct

6.2.2 AIM3 선형층 생성 함수 AIM3_GenerateLinear

AIM3 선형층 생성 함수 AIM3_GenerateLinear(알고리즘 4)는 λ비트 길이의 비트열 iv를 입력으로 받아, 임의의 이진 행렬 $(A_0, \dots, A_{2\ell}) \in (\{0,1\}^{n\times n})^{2\ell}$과 길이 n의 이진 벡터 $(b_0, \dots, b_{\ell+1}) \in (\{0,1\}^n)^{2\ell+1}$ 로 구성된 선형층을 생성하여 출력한다.

확장 출력 함수 SHAKE_λ에 비트열 iv와 선형층 구성에 필요한 비트 길이 $2\ell n^2 + (\ell+1)n$를 입력하여 $tape$를 생성한다. 임의의 이진 행렬을 생성하기 위해, 상삼각행렬(upper triangular matrix)과 하삼각행렬(lower triangular matrix)을 구성한다. 이 때, 대각 성분은 모두 1로 설정하고, 대각 성분을 제외한 성분은 $tape$에서 가져온 값 또는 0으로 채운다. 생성된 상삼각행렬과 하삼각형렬의 곱셈 결과를 이진 행렬로 설정한다.

AIM3_GenerateLinear 함수는 키 생성 과정, 서명 과정, 검증 과정에서 사용된다.

비고 AIM3_GenerateLinear 함수에서 행렬을 생성할 때, 행과 열이 바뀐 $L[column][row]$ 형식으로 저장되지만, 생성된 행렬을 다른 함수에서 사용 시 항상 $L[row][column]$ 형식으로 접근한다.

알고리즘 4 AIM3_GenerateLinear(iv) — AIM3 이진 행렬과 벡터 생성

입력: 비트열 $iv \in \{0,1\}^\lambda$
출력: 이진 행렬 $(A_j)_{j\in[2\ell]} \in (\{0,1\}^{n\times n})^{2\ell}$, 이진 벡터 $(b_j)_{j\in[\ell+1]} \in (\{0,1\}^n)^{\ell+1}$

1: **for** j **from** 0 **to** $(2\ell-1)$ **do**

2: $L_j \leftarrow 0^{n\times n},\ U_j \leftarrow 0^{n\times n}$

3: **end for**

4: **for** j **from** 0 **to** ℓ **do**

5: $b_j \leftarrow 0^n$

6: **end for**

7: $tape \leftarrow \mathrm{SHAKE}_\lambda(iv, 2\ell n^2 + (\ell+1)n)$ ▷ **8.1** 참조

8: **for** j **from** 0 **to** $(2\ell-1)$ **do**

9: **for** r **from** 0 **to** $(n-1)$ **do**

10: **for** c **from** 0 **to** $(n-1)$ **do**

11: **if** $c < r$ **then**

12: $L_j[c][r] \leftarrow tape[jn^2 + rn + c]$

13: $U_j[c][r] \leftarrow 0$

14: **else if** $c = r$ **then**

15: $L_j[c][r] \leftarrow 1$

16: $U_j[c][r] \leftarrow 1$

17: **else**

18: $U_j[c][r] \leftarrow tape[jn^2 + rn + c]$

19: $L_j[c][r] \leftarrow 0$

20: **end if**

21: **end for**

22: **end for**

23: **end for**

24: **for** j **from** 0 **to** $(2\ell - 1)$ **do**

25: **for** c **from** 0 **to** $(n - 1)$ **do**

26: $A_j[c] \leftarrow \mathrm{MatVecMul}(U_j, L_j[c])$ ▷ **8.2.3** 및 알고리즘 17 참조

27: **end for**

28: **end for**

29: **for** j **from** 0 **to** ℓ **do**

30: **for** r **from** 0 **to** $(n - 1)$ **do**

31: $b_j[r] \leftarrow tape[2\ell n^2 + (j-1)n + r]$

32: **end for**

33: **end for**

34: **return** $\left(\left(A_j\right)_{j\in[2\ell]}, \left(b_j\right)_{j\in[l+1]}\right)$

6.3 AIM3 응용 함수

6.3.1 AIM3 S-box 계산 함수 AIM3_SboxOutputs

AIM3 S-box 계산 함수 AIM3_SboxOutputs (알고리즘 5)는 이진 행렬과 비트열의 배열 $L = \left(\left(A_j\right)_{j\in[2\ell]}, \left(b_j\right)_{j\in[\ell+1]}\right) \in (\{0,1\}^{n\times n})^{2\ell} \times (\{0,1\}^n)^{\ell+1}$, n비트 길이의 비트열 pt, n비트 길이의 비트열 ct를 입력으로 받아 S-box 계산까지 수행하고, 그 결과 값을 비트열의 배열 $(y_0, \ldots, y_\ell) \in (\{0,1\}^n)^{\ell+1}$로 구성하여 출력한다.

AIM3_SboxOutputs 함수는 서명 과정에서 사용된다.

알고리즘 5 $\mathrm{AIM3\_SboxOutputs}\left(L = \left(\left(A_j\right)_{j\in[2\ell]}, \left(b_j\right)_{j\in[\ell+1]}\right), pt, ct\right)$ — AIM3 S-box 계산

입력: 이진 행렬과 비트열의 배열 $L = \left(\left(A_j\right)_{j\in[2\ell]}, \left(b_j\right)_{j\in[\ell+1]}\right) \in (\{0,1\}^{n\times n})^{2\ell} \times (\{0,1\}^n)^{\ell+1}$, 비트열 $pt \in \{0,1\}^n$, 비트열 $ct \in \{0,1\}^n$
출력: 비트열의 배열 $\left(y_j\right)_{j\in[\ell+1]} \in (\{0,1\}^n)^{\ell+1}$

1: **for** j **from** 0 **to** $\ell - 1$ **do**
2: $\quad x_j \leftarrow \mathrm{MatVecMul}\left(A_j, pt\right) + b_j$ ▷ **8.2.3** 및 알고리즘 17 참조
3: $\quad y_i \leftarrow \mathrm{Exp}\left(x_j, 2^{e_j} - 1\right) + \mathrm{Exp}\left(x_j, 2^n - 2\right)$ ▷ **8.2.2** 및 알고리즘 16 참조
4: **end for**
5: $y_\ell \leftarrow pt + ct$
6: **return** $\left(y_j\right)_{j\in[\ell+1]}$

6.3.2 AIM3 다자간 계산 도움 함수 AIM3_MPC

AIM3 다자간 계산 도움 함수 AIM3_MPC (알고리즘 6)는 이진 행렬과 비트열의 배열 $L = \left(\left(A_j\right)_{j\in[2\ell]}, \left(b_j\right)_{j\in[\ell+1]}\right) \in (\{0,1\}^{n\times n})^{2\ell} \times (\{0,1\}^n)^{\ell+1}$, n비트 길이의 비트열 ct, n비트 길이의 비트열 $pt^{(\cdot)}$, 비트열의 배열 $\left(y_0^{(\cdot)}, \ldots, y_{\ell-1}^{(\cdot)}\right) \in (\{0,1\}^n)^\ell$, 정수 $i \in [N]$ 를 입력으로 받아, 비트열의 배열 $(x_0^{(\cdot)}, \ldots, x_\ell^{(\cdot)}, y_0^{(\cdot)}, \ldots, y_\ell^{(\cdot)}, z_0^{(\cdot)}, \ldots, z_\ell^{(\cdot)}) \in (\{0,1\}^n)^{3(\ell+1)}$을 출력한다. AIM3_MPC 함수는 AIM3의 S-box를 직접 계산하지 않고 곱셈을 검증하기 위하여 S-box의 입력과 출력의 곱을 계산하여 출력한다.

AIM3_MPC 함수는 서명 과정과 검증 과정에서 사용된다.

알고리즘 6 $\mathrm{AIM3\_MPC}\left(L = \left(\left(A_j\right)_{j\in[2\ell]}, \left(b_j\right)_{j\in[\ell+1]}\right), ct, pt^{(\cdot)}, \left(y_j^{(\cdot)}\right)_{j\in[\ell]}, i\right)$ — AIM3 다자간 계산 도움 함수

입력: 이진 행렬과 비트열의 배열 $L = \left(\left(A_j\right)_{j\in[2\ell]}, \left(b_j\right)_{j\in[\ell+1]}\right) \in (\{0,1\}^{n\times n})^{2\ell} \times (\{0,1\}^n)^{\ell+1}$, 비트열 $ct \in \{0,1\}^n$, 비트열 $pt^{(\cdot)} \in \{0,1\}^n$, 비트열의 배열 $\left(y_j^{(\cdot)}\right)_{j\in[\ell]} \in (\{0,1\}^n)^\ell$, 정수 $i \in [N]$
출력: 비트열의 배열 $\left(x_j^{(\cdot)}, y_j^{(\cdot)}, z_j^{(\cdot)}\right)_{j\in[\ell+1]} \in (\{0,1\}^n)^{3(\ell+1)}$

1: **for** j **from** 0 **to** $(\ell - 1)$ **do**
2: $\quad x_j^{(\cdot)} \leftarrow \mathrm{MatVecMul}\left(A_j, pt^{(\cdot)}\right)$ ▷ **8.2.3** 및 알고리즘 17 참조
3: **end for**
4: $x_\ell^{(\cdot)} \leftarrow \sum_{j\in[\ell]} \mathrm{MatVecMul}\left(A_{j+\ell}, y_j^{(\cdot)}\right)$ ▷ **8.2.3** 및 알고리즘 17 참조
5: **if** $i = N - 1$ **then**
6: $\quad$ **for** j **from** 0 **to** ℓ **do**
7: $\quad\quad x_j^{(\cdot)} \leftarrow x_j^{(\cdot)} + b_j$
8: $\quad$ **end for**
9: $\quad y_\ell^{(\cdot)} \leftarrow y_\ell^{(\cdot)} \oplus ct$
10: **end if**
11: **for** j **from** 0 **to** ℓ **do**
12: $\quad z_j^{(\cdot)} \leftarrow \mathrm{Exp}\left(x_j^{(\cdot)}, 2^{e_j}\right) + \mathrm{IntToBits}(1, n)$ ▷ **5.1.1**, **8.2.2** 및 알고리즘 1, 16 참조

13: **end for**

14: **return** $\left(x_j^{(\cdot)}, y_j^{(\cdot)}, z_j^{(\cdot)}\right)_{j\in[\ell+1]}$

7 AIMer 매개변수

디지털 서명 알고리즘 AIMer에서 사용하는 매개변수 6종은 **표 4**에 정리되어 있다. 매개변수는 AIM3 매개변수$(n, \ell, e_0, e_1, e_2, e_3)$, 영지식증명 매개변수$(\lambda, N, \tau)$ 및 해시 함수 H_0의 접두사 $prefix$로 구성된다.

비고 각 세부 알고리즘 이름의 숫자는 보안 매개변수 λ를 나타내고, 알파벳 f와 s는 알고리즘의 특성을 구분한다. f는 서명 과정과 검증 과정의 속도를 중시한 알고리즘이며, s는 서명의 크기가 작은 알고리즘이다. 보안 매개변수 λ가 동일한 경우, f와 s 알고리즘은 동일한 AIM3 매개변수를 사용하며, 영지식증명 매개변수인 가상참여자 수 N과 반복 횟수 τ의 값을 조정하여 알고리즘의 특성을 구분한다.

표 4 — AIMer 매개변수

구분	λ	n	N	τ	ℓ	e_0	e_1	e_2	e_3	$prefix$
AIMer-128f	128	128	16	33	2	3	7	11	–	0x00
AIMer-128s	128	128	256	17	2	3	7	11	–	0x10
AIMer-192f	192	192	16	49	2	11	23	47	–	0x20
AIMer-192s	192	192	256	25	2	11	23	47	–	0x30
AIMer-256f	256	256	16	65	3	3	7	11	19	0x40
AIMer-256s	256	256	256	33	3	3	7	11	19	0x50

8 보조 함수

8.1 해시 함수와 확장 함수

이 표준에서는 ISO/IEC 10118-3:2018에서 규정된 확장 출력 함수 SHAKE-128과 SHAKE-256를 사용하여야 한다. 확장 출력 함수는 비트열 M과 출력할 비트 길이 d를 입력으로 받아 비트열 $output$을 출력하며 다음과 같이 표기한다.

$$output \leftarrow \text{SHAKE-128}(M, d) \text{ 또는 } output \leftarrow \text{SHAKE-256}(M, d)$$

이 표준에서는 보안 매개변수 $\lambda = 128$인 경우 SHAKE-128, $\lambda = 192$ 또는 256인 경우 SHAKE-256 함수를 사용하며, 이를 확장 출력 함수 SHAKE_λ로 정의한다.

$$\text{SHAKE}_\lambda(M, d) = \begin{cases} \text{SHAKE-128}(M, d) \textbf{ if } \lambda = 128 \\ \text{SHAKE-256}(M, d) \textbf{ if } \lambda = 192, 256 \end{cases}$$

확장 출력 함수 SHAKE_λ를 이용하여 이 표준에서 사용하는 해시 함수와 확장 함수를 정의한다. 확장 함수는 내부적으로 확장 출력 함수를 사용하지만, 해시 함수와는 달리 정수 값을 출력할 수 있다. 해시 함수 또는 확장 함수에 비트열을 입력할 때 다음 규칙을 적용한다.

- 0에서 255 사이의 정수를 입력할 때, IntToBits 함수(**5.1.1** 및 알고리즘 1 참조)를 사용하여 길이가 8인 비트열로 변환한 후 입력한다.
- 해시 함수는 도메인 구분을 위해 접두사(prefix)를 입력의 앞부분에 배치한다.
- 인덱스를 가진 리스트나 벡터 등을 입력이나 출력으로 사용할 경우, 인덱스 0부터 사용한다.

해시 함수와 확장 함수의 작동방식은 알고리즘 7부터 알고리즘 14에 제시한다.

H_0	접두사 0xa0, 2λ비트 출력 a값은 AIMer의 매개변수별 아래와 같이 변경하여 사용 128f: 0, 128s: 1, 192f: 2, 192s: 3, 256f: 4, 256s:5 (**표 4**의 $prefix$ 참조)
H_1	접두사 0x01, 2λ비트 출력
H_2	접두사 0x02, 2λ비트 출력
H_3	접두사 0x03, $(\tau+1)\lambda$비트 출력
H_4	접두사 0x04, 2λ비트 출력
H_5	접두사 0x05, $(2\ell+5)\lambda$비트 출력
ExpandH1	접두사 없음, $(\ell+1)\tau\lambda$비트 출력
ExpandH2	접두사 없음, τ 길이의 정수 배열 출력, 각 정수는 0부터 255 사이의 값을 가짐

8.1.1 해시 함수 H_0

해시 함수 H_0(알고리즘 7)는 λ비트 길이의 비트열 iv, λ비트 길이의 비트열 ct, 비트열 M'을 입력으로 받는다. 해시 함수 H_0는 AIMer 매개변수에 따라 도메인을 구분하기 위한 접두사 $prefix$ 값을 변경하여 사용한다(**7**절 및 **표 4** 참조). IntToBits 함수(**5.1.1** 및 알고리즘 1 참조)를 사용하여 매개변수별 접두사 $prefix$를 8비트 길이의 비트열로 변환하고, 입력으로 받은 비트열 iv, ct, M'과 순서대로 연접한 값을 확장 출력 함수 SHAKE_λ에 입력하여, 2λ비트 길이의 비트열 μ를 출력한다.

알고리즘 7 $\mathrm{H}_0(iv, ct, M')$ — 해시 함수 H_0

입력: 비트열 $iv \in \{0,1\}^\lambda$, 비트열 $ct \in \{0,1\}^\lambda$, 비트열 $M' \in \{0,1\}^*$
출력: 비트열 $\mu \in \{0,1\}^{2\lambda}$

1: $x \leftarrow \mathrm{IntToBits}(prefix, 8) \,\|\, iv \,\|\, ct \,\|\, M'$ ▷ **5.1.1** 및 알고리즘 1 참조
2: $\mu \leftarrow \mathrm{SHAKE}_\lambda(x, 2\lambda)$ ▷ **8.1** 참조
3: **return** μ

8.1.2 해시 함수 H_1

해시 함수 H_1(알고리즘 8)은 2λ비트 길이의 비트열 μ, λ비트 길이의 비트열 $salt$, $2\lambda \times N \times \tau$비트 길이의 비트열 배열 $\left((com_k^{(i)})_{i\in[N]}\right)_{k\in[\tau]}$, $n \times \tau$비트 길이의 비트열 배열 $(\Delta pt_k)_{k\in[\tau]}$, $n \times \ell \times \tau$비트 길이의 비트열 배열 $\left((\Delta y_{k,j})_{j\in[\ell]}\right)_{k\in[\tau]}$, $n \times \tau$비트 길이의 비트열 배열 $(\Delta c_k)_{k\in[\tau]}$을 입력으로 받는다. IntToBits 함수(**5.1.1** 및 알고리즘 1 참조)를 사용하여 접두사 0x01를 8비트 길이의 비트열로 변환하고, 입력으로 받은 비트열과 연접한 값을 확장 출력 함수 SHAKE_λ에 입력하여, 2λ비트 길이의 해시 값 h_1를 출력한다.

알고리즘 8 $\mathrm{H}_1\left(\mu, salt, \left((com_k^{(i)})_{i\in[N]}, \Delta pt_k, (\Delta y_{k,j})_{j\in[\ell]}, \Delta c_k\right)_{k\in[\tau]}\right)$ — 해시 함수 H_1

입력: 비트열 $\mu \in \{0,1\}^{2\lambda}$, 비트열 $salt \in \{0,1\}^\lambda$, 비트열의 배열 $\left((com_k^{(i)})_{i\in[N]}\right)_{k\in[\tau]} \in \left(\{0,1\}^{2\lambda}\right)^{\tau N}$, 비트열의 배열 $(\Delta pt_k)_{k\in[\tau]} \in (\{0,1\}^n)^\tau$, 비트열의 배열 $\left((\Delta y_{k,j})_{j\in[\ell]}\right)_{k\in[\tau]} \in (\{0,1\}^n)^{\tau\ell}$, 비트열의 배열 $(\Delta c_k)_{k\in[\tau]} \in (\{0,1\}^n)^\tau$
출력: 비트열 $h_1 \in \{0,1\}^{2\lambda}$

1: $x \leftarrow \text{IntToBits}(\texttt{0x01},8) \parallel \mu \parallel salt$ ▷ **5.1.1** 및 알고리즘 1 참조
2: **for** k **from** 0 **to** $(\tau - 1)$ **do**
3: **for** i **from** 0 **to** $(N - 1)$ **do**
4: $x \leftarrow x \parallel com_k^{(i)}$
5: **end for**
6: $x \leftarrow x \parallel \Delta pt_k$
7: **for** j **from** 0 **to** $(\ell - 1)$ **do**
8: $x \leftarrow x \parallel \Delta y_{k,j}$
9: **end for**
10: $x \leftarrow x \parallel \Delta c_k$
11: **end for**
12: $h_1 \leftarrow \text{SHAKE}_\lambda(x, 2\lambda)$ ▷ **8.1** 참조
13: **return** h_1

8.1.3 해시 함수 H_2

해시 함수 H_2(알고리즘 9)는 2λ비트 길이의 비트열 h_1, λ비트 길이의 비트열 $salt$, $n \times \ell \times N \times \tau$비트 길이의 비트열 배열 $\left((\alpha_{k,0}^{(i)}, \alpha_{k,1}^{(i)}, \ldots, \alpha_{k,\ell}^{(i)})_{i\in[N]}\right)_{k\in[\tau]}$, $n \times N \times \tau$비트 길이의 비트열 배열 $\left((v_k^{(i)})_{i\in[N]}\right)_{k\in[\tau]}$을 입력으로 받는다. IntToBits 함수(**5.1.1** 및 알고리즘 1 참조)를 사용하여 접두사 `0x02`를 8비트 길이의 비트열로 변환하고, 입력으로 받은 비트열과 연접한 값을 확장 출력 함수 SHAKE_λ에 입력하여, 2λ비트 길이의 해시 값 h_2를 출력한다.

알고리즘 9 $\text{H}_2\left(h_1, salt, \left((\alpha_{k,0}^{(i)}, \alpha_{k,1}^{(i)}, \ldots, \alpha_{k,\ell}^{(i)})_{i\in[N]}, (v_k^{(i)})_{i\in[N]}\right)_{k\in[\tau]}\right)$ — 해시 함수 H_2

입력: 비트열 $h_1 \in \{0,1\}^{2\lambda}$, 비트열 $salt \in \{0,1\}^{\lambda}$, 비트열의 배열 $\left((\alpha_{k,0}^{(i)}, \alpha_{k,1}^{(i)}, \ldots, \alpha_{k,\ell}^{(i)})_{i\in[N]}\right)_{k\in[\tau]} \in (\{0,1\}^n)^{N\tau}$, 비트열의 배열 $\left((v_k^{(i)})_{i\in[N]}\right)_{k\in[\tau]} \in (\{0,1\}^n)^{N\tau}$
출력: 비트열 $h_2 \in \{0,1\}^{2\lambda}$

1: $x \leftarrow \text{IntToBits}(\texttt{0x02},8) \parallel h_1 \parallel salt$ ▷ **5.1.1** 및 알고리즘 1 참조
2: **for** k **from** 0 **to** $(\tau - 1)$ **do**
3: **for** i **from** 0 **to** $(N - 1)$ **do**
3: **for** j **from** 0 **to** ℓ **do**
4: $x \leftarrow x \parallel \alpha_{k,j}^{(i)}$
5: **end for**
6: **for** i **from** 0 **to** $(N - 1)$ **do**
7: $x \leftarrow x \parallel v_k^{(i)}$
8: **end for**
9: **end for**

10: $h_2 \leftarrow \mathrm{SHAKE}_\lambda(x, 2\lambda)$ ▷ **8.1** 참조

11: **return** h_2

8.1.4 해시 함수 H_3

해시 함수 H_3(알고리즘 10)는 2λ비트 길이의 비트열 μ, λ비트 길이의 비트열 pt, λ비트 길이의 비트열 ρ을 입력으로 받는다. IntToBits 함수(**5.1.1** 및 알고리즘 1 참조)를 사용하여 접두사 `0x03`를 8비트 길이의 비트열로 변환하고, 입력으로 받은 비트열과 연접한 값을 확장 출력 함수 SHAKE_λ에 입력하여, $(\tau+1)\lambda$비트 길이의 비트열 buf를 생성한다. 비트열 buf의 처음 λ비트를 비트열 $salt$로 설정하고, 나머지 $\tau\lambda$비트를 λ비트 단위로 분할하여 τ개의 비트열의 배열 $(seed_i)_{i\in[\tau]}$로 설정하여 출력한다.

알고리즘 10 $\mathrm{H}_3(\mu, pt, \rho)$ — 해시 함수 H_3

입력: 비트열 $\mu \in \{0,1\}^{2\lambda}$, 비트열 $pt \in \{0,1\}^{\lambda}$, 비트열 $\rho \in \{0,1\}^{\lambda}$
출력: 비트열의 배열 $\left(salt, (seed_i)_{i\in[\tau]}\right) \in \{0,1\}^{\lambda} \times \{0,1\}^{\lambda\tau}$

1: $x \leftarrow \mathrm{IntToBits}(\texttt{0x03},8) \| \mu \| pt \| \rho$ ▷ **5.1.1** 및 알고리즘 1 참조

2: $buf \leftarrow \mathrm{SHAKE}_\lambda(x, (\tau+1)\lambda)$ ▷ **8.1** 참조

3: $salt \leftarrow buf[0\colon\lambda]_{\mathrm{bit}}$

4: **for** i **from** 0 **to** $\tau - 1$ **do**

5: $\quad seed[i] \leftarrow buf[(i+1)\lambda\colon(i+2)\lambda]_{\mathrm{bit}}$

6: **end for**

7: **return** $\left(salt, (seed_i)_{i\in[\tau]}\right)$

8.1.5 해시 함수 H_4

해시 함수 H_4(알고리즘 11)는 λ비트 길이의 비트열 $salt$, 음이 아닌 정수 $k \in [\tau]$, 음이 아닌 정수 $i \in [N]$, λ비트 길이의 비트열 $node$을 입력으로 받는다. 접두사 `0x04`를 IntToBits 함수(**5.1.1** 및 알고리즘 1 참조)를 사용하여 8비트 길이의 비트열로 변환한 값, 비트열 $salt$, 정수 k와 i를 각각 IntToBits 함수를 사용하여 8비트 길이의 비트열로 변환한 값, 비트열 $node$를 순서대로 연접한 값을 확장 출력 함수 SHAKE_λ에 입력하여, 2λ비트 길이의 비트열 buf를 생성한다. 비트열 buf의 처음 λ비트를 비트열 $left$로 설정하고, 나머지 λ비트를 비트열 $right$로 설정하여 출력한다.

알고리즘 11 $\mathrm{H}_4(salt, k, i, node)$ — 해시 함수 H_4

입력: 비트열 $salt \in \{0,1\}^{\lambda}$, 정수 $k \in [\tau]$, 정수 $i \in [N]$, 비트열 $node \in \{0,1\}^{\lambda}$
출력: 비트열의 배열 $(left, right) \in \{0,1\}^{\lambda} \times \{0,1\}^{\lambda}$

1: $x \leftarrow \mathrm{IntToBits}(\texttt{0x04},8) \| salt \| \mathrm{IntToBits}(k,8) \| \mathrm{IntToBits}(i,8) \| node$ ▷ **5.1.1** 및 알고리즘 1 참조

2: $buf \leftarrow \mathrm{SHAKE}_\lambda(x, 2\lambda)$ ▷ **8.1** 참조

3: $left \leftarrow buf[0\colon\lambda]_{\mathrm{bit}}$

4: $right \leftarrow buf[\lambda\colon 2\lambda]_{\mathrm{bit}}$

5: **return** $(left, right)$

8.1.6 해시 함수 H_5

해시 함수 H_5(알고리즘 12)는 λ비트 길이의 비트열 $salt$, 음이 아닌 정수 $k \in [\tau]$, 음이 아닌 정수 $i \in [N]$, λ비트 길이의 비트열 $seed$를 입력으로 받는다. 접두사 `0x05`를 IntToBits 함수(**5.1.1** 및 알고리즘 1 참조)를 사용하여 8비트 길이의 비트열로 변환한 값, 비트열 $salt$, 정수 k와 i를 각각 IntToBits 함수를 사용하여 8비트 길이의 비트열로 변환한 값, 비트열 $seed$를 순서대로 연접한 값을 확장 출력 함수 SHAKE_λ에 입력하여, $(2\ell+5)\lambda$비트 길이의 비트열 buf를 생성한다. 비트열 buf는 인덱스 순서대로 2λ비트를 비트열 $commit$, λ비트를 비트열 pt, $\lambda\ell$비트를 각 비트열이 λ비트 길이인 비트열 배열 $(y_j)_{j\in[\ell]}$, $(\ell+1)\lambda$비트를 각 비트열이 λ비트 길이인 비트열 배열 $(a_j)_{j\in[\ell+1]}$, λ비트를 비트열 c로 설정하여 출력한다.

알고리즘 12 $\mathrm{H}_5(salt, k, i, seed)$ — 해시 함수 H_5

입력: 비트열 $salt \in \{0,1\}^\lambda$, 정수 $k \in [\tau]$, 정수 $i \in [N]$, 비트열 $seed \in \{0,1\}^\lambda$

출력: 비트열 $commit \in \{0,1\}^{2\lambda}$, 비트열 $pt \in \{0,1\}^\lambda$, 비트열의 배열 $(y_j)_{j\in[\ell]} \in (\{0,1\}^\lambda)^\ell$, 비트열의 배열 $(a_j)_{j\in[\ell+1]} \in (\{0,1\}^\lambda)^{\ell+1}$, 비트열 $c \in \{0,1\}^\lambda$

$x \leftarrow \mathrm{IntToBits}(\texttt{0x05},8) \,\|\, salt \,\|\, \mathrm{IntToBits}(k,8) \,\|\, \mathrm{IntToBits}(i,8) \,\|\, seed$ ▷ **5.1.1** 및 알고리즘 1 참조

$buf \leftarrow \mathrm{SHAKE}_\lambda(x, (2\ell+5)\lambda)$ ▷ **8.1** 참조

$commit \leftarrow buf[0\colon 2\lambda]_{\mathrm{bit}}$

$pt \leftarrow buf[2\lambda\colon 3\lambda]_{\mathrm{bit}}$

for j **from** 0 **to** $(\ell-1)$ **do**

    $y_j \leftarrow buf[(j+3)\lambda\colon (j+4)\lambda]_{\mathrm{bit}}$

end for

for j **from** 0 **to** ℓ **do**

    $a_j \leftarrow buf[(j+3+\ell)\lambda\colon (j+4+\ell)\lambda]_{\mathrm{bit}}$

end for

$c \leftarrow buf[(2\ell+4)\lambda\colon (2\ell+5)\lambda]_{\mathrm{bit}}$

return $\left(commit, pt, (y_j)_{j\in[\ell]}, (a_j)_{j\in[\ell+1]}, c\right)$

8.1.7 확장 함수 ExpandH1

확장 함수 ExpandH1(알고리즘 13)는 2λ비트 길이의 비트열 h_1을 확장 출력 함수 SHAKE_λ에 입력하여, $(\ell+1)\tau\lambda$비트 길이의 비트열 buf를 생성한다. 비트열 buf는 λ비트 단위로 분할하여 $(\ell+1)\tau$개의 비트열의 배열 $\left((\epsilon_{k,j})_{j\in[\ell+1]}\right)_{k\in[\tau]}$로 설정한다.

알고리즘 13 $\mathrm{ExpandH1}(h_1)$ — 해시 함수 ExpandH1

입력: 비트열 $h_1 \in \{0,1\}^{2\lambda}$

출력: 비트열의 배열 $\left((\epsilon_{k,j})_{j\in[\ell+1]}\right)_{k\in[\tau]} \in (\{0,1\}^\lambda)^{\tau(\ell+1)}$

$buf \leftarrow \mathrm{SHAKE}_\lambda(h_1, (\ell+1)\tau\lambda)$ ▷ **8.1** 참조

2: **for** k **from** 0 **to** $\tau - 1$ **do**

3: **for** j **from** 0 **to** ℓ **do**

4: $\epsilon_{k,j} \leftarrow buf[(\ell+1)k\lambda + j\lambda : (\ell+1)k\lambda + (j+1)\lambda]_{\text{bit}}$

5: **end for**

6: **end for**

7: **return** $\left((\epsilon_{k,j})_{j\in[\ell+1]}\right)_{k\in[\tau]}$

8.1.8 확장 함수 ExpandH2

확장 함수 ExpandH2(알고리즘 14)는 2λ비트 길이의 비트열 h_2을 확장 출력 함수 SHAKE_λ에 입력하여, 8τ비트 길이의 비트열 buf를 생성한다. 비트열 buf는 8비트 단위로 분할하여 BitsToInt 함수(**5.1.1** 및 알고리즘 2 참조)에 입력하여 정수로 변환하고, $\bmod N$ 연산을 수행하여 N보다 작은 정수 τ개를 출력한다.

알고리즘 14 ExpandH2(h_2) — 해시 함수 ExpandH2

입력: 비트열 $h_2 \in \{0,1\}^{2\lambda}$
출력: 음이 아닌 정수의 배열 $(\bar{i}_0, \ldots, \bar{i}_{\tau-1}) \in ([N])^\tau$

1: $buf \leftarrow \text{SHAKE}_\lambda(h_2, 8\tau)$ ▷ **8.1** 참조

2: **for** k **from** 0 **to** $\tau - 1$ **do**

3: $\bar{i}_k \leftarrow \text{BitsToInt}(buf[8k : 8(k+1)]_{\text{bit}}, 8)$ ▷ **5.1.1** 및 알고리즘 2 참조

4: $\bar{i}_k \leftarrow \bar{i}_k \bmod N$

5: **end for**

6: **return** $(\bar{i}_0, \ldots, \bar{i}_{\tau-1})$

8.2 유한체 연산 함수

AIMer 알고리즘은 보안 매개변수 $\lambda = n$에 따라 유한체 $\text{GF}(2^{128})$, $\text{GF}(2^{192})$, $\text{GF}(2^{256})$를 사용한다. 각 유한체를 정의하기 위해 사용되는 차수 n의 기약다항식은 다음과 같다.

- $\text{GF}(2^{128})$: $f(X) = X^{128} + X^7 + X^2 + X + 1$.
- $\text{GF}(2^{192})$: $f(X) = X^{192} + X^7 + X^2 + X + 1$.
- $\text{GF}(2^{256})$: $f(X) = X^{256} + X^{10} + X^5 + X^2 + 1$.

기약다항식 $f(X)$의 최고차항 X^n을 제외한 나머지 항을 정수로 변환하여, 유한체 곱셈 함수의 감산 상수로 사용한다. $\text{GF}(2^{128})$ 및 $\text{GF}(2^{192})$에서는 `0x87`, $\text{GF}(2^{256})$에서는 `0x425`를 감산 상수로 사용한다.

8.2.1 유한체 곱셈 함수 Mul

유한체 곱셈 함수 Mul(알고리즘 15)는 n비트 길이의 비트열 x와 y를 입력으로 받아 n비트 길이의 비트열 $z = x \cdot y$를 출력한다.

알고리즘 15 $\mathrm{Mul}(x, y)$ — 유한체 곱셈 함수

입력: 비트열 $x \in \{0,1\}^n$, 비트열 $y \in \{0,1\}^n$
출력: 비트열 $z = x \cdot y \in \{0,1\}^n$

$z \leftarrow 0^n$
$v \leftarrow x$
if $n = 128$ **or** $n = 192$ **then**
    $r \leftarrow \mathrm{IntToBits}(\texttt{0x87}, n)$ ▷ **5.1.1** 및 알고리즘 1 참조
else if $n = 256$ **then**
    $r \leftarrow \mathrm{IntToBits}(\texttt{0x425}, n)$ ▷ **5.1.1** 및 알고리즘 1 참조
end if
for i **from** 0 **to** $n - 1$ **do**
    if $y[i] = 1$ **then**
        $z \leftarrow z \oplus v$
    end if
    $msb \leftarrow v[n-1]$
    $v \leftarrow v \ll 1$ ▷ 비트 시프트 후 하위 n비트만 유지
    if $msb = 1$ **then**
        $v \leftarrow v \oplus r$
    end if
end for
return z

8.2.2 유한체 거듭제곱 함수 Exp

유한체 거듭제곱 함수 Exp(알고리즘 16)는 n비트 길이의 비트열 x와 음이 아닌 정수 e를 입력으로 받아 n비트 길이의 비트열 $z = x^e$를 계산하여 출력한다.

알고리즘 16 $\mathrm{Exp}(x, e)$ — 유한체 거듭제곱 함수

입력: 비트열 $x \in \{0,1\}^n$, 음이 아닌 정수 e
출력: 비트열 $z = x^e \in \{0,1\}^n$

$z \leftarrow 1$
$e' \leftarrow \mathrm{IntToBits}(e, n)$ ▷ **5.1.1** 및 알고리즘 1 참조
for i **from** 0 **to** $n - 1$ **do**
    if $e'[i] = 1$ **then**
        $z \leftarrow \mathrm{Mul}(z, x)$ ▷ **8.2.1** 및 알고리즘 15 참조
    end if

7: $x \leftarrow \mathrm{Mul}(x, x)$ ▷ **8.2.1** 및 알고리즘 15 참조

8: **end for**

9: **return** z

8.2.3 행렬-벡터 곱셈 함수 MatVecMul

이진 행렬과 벡터 곱셈 함수 MatVecMul(알고리즘 17)은 $n \times n$ 이진 행렬 A와 n비트 길이의 비트열 x를 입력으로 받아 n비트 길이의 비트열 $z = A \cdot x$를 출력한다.

알고리즘 17 $\mathrm{MatVecMul}(A, x)$ — 행렬-벡터 곱셈 함수

입력: 이진 행렬 $A \in \{0,1\}^{n \times n}$, 비트열 $x \in \{0,1\}^n$
출력: 비트열 $z \in \{0,1\}^n$

1: $z \leftarrow 0^n$

2: **for** i **from** 0 **to** $n-1$ **do**

3: **if** $x[i] = 1$ **then**

4: **for** j **from** 0 **to** $n-1$ **do**

5: $z[j] \leftarrow z[j] \oplus A[i][j]$

6: **end for**

7: **end if**

8: **end for**

9: **return** z

8.3 GGM 트리 함수

AIMer 알고리즘은 GGM(Goldreich-Goldwasser-Micali) 트리[2] 구조를 사용한다. GGM 트리는 포화 이진 트리 구조를 기반으로 하여 N개의 의사난수 값 중 하나를 제외한 나머지 $N-1$개의 의사난수 값을 효율적으로 재구성할 수 있는 성질을 가지고 있다.

8.3.1 GGM 트리 확장 함수 ExpandTree

GGM 트리 확장 함수 ExpandTree(알고리즘 18)는 λ비트 길이의 비트열 $salt$, 정수 $k \in [\tau]$, λ비트 길이의 비트열 $seed$를 입력으로 받아, N개의 단말 노드를 갖는 GGM 트리 $nodes$로 확장하여 출력한다. 확장된 GGM 트리 $nodes$는 총 $2N-1$개의 노드로 구성되며, 각 노드의 값은 해시 함수 H_4(**8.1.5** 및 알고리즘 11 참조)를 사용하여 계산한다. 모든 노드의 값은 λ비트 길이의 비트열이다.

ExpandTree 함수는 서명 과정에서 사용한다.

알고리즘 18 $\mathrm{ExpandTree}(salt, k, seed)$ — GGM 트리 확장 함수

입력: 비트열 $salt \in \{0,1\}^\lambda$, 정수 $k \in [\tau]$, 비트열 $seed \in \{0,1\}^n$
출력: 트리 $nodes[0:2N-1] \in (\{0,1\}^\lambda)^{2N-1}$

1: $nodes[0:2N-1] \leftarrow \left(0^\lambda\right)^{2N-1}$

2: $nodes[0] \leftarrow seed$

3: **for** i **from** 1 **to** $(N-1)$ **do**

4: $\quad (nodes[2i-1], nodes[2i]) \leftarrow \mathrm{H}_4(salt, k, i, nodes[i-1])$ ▷ **8.1.5** 및 알고리즘 11 참조

5: **end for**

6: **return** $nodes[0:2N-1]$

8.3.2 공개 경로 생성 함수 RevealAllBut

공개 경로(revealing path) 생성 함수 RevealAllBut(알고리즘 19)는 GGM 트리 $nodes$와 비공개 인덱스 $\bar{i}$를 입력으로 받는다. GGM 트리의 N개의 단말 노드 중 비공개 인덱스에 대응하는 단말 노드를 제외한 $N-1$개의 단말 노드에 대한 공개 경로를 생성한다. 공개 경로 $path$는 GGM 트리의 비공개 단말 노드에서 루트 노드에 이르는 $\log N$개의 형제 노드의 값으로 구성하여 출력한다. 영지식증명 매개변수 N은 16 또는 256을 사용하기 때문에 $\log N$은 4 또는 8이다.

RevealAllBut 함수는 서명 과정에서 사용한다.

알고리즘 19 RevealAllBut$(nodes, \bar{i})$ — 공개 경로 생성 함수

입력: 트리 $nodes[0:2N-1] \in (\{0,1\}^\lambda)^{2N-1}$, 비공개 인덱스 $\bar{i} \in [N]$
출력: 공개 경로 $path[0:\log N] \in (\{0,1\}^\lambda)^{\log N}$

1: $path[0:\log N] \leftarrow \left(0^\lambda\right)^{\log N}$

2: $j \leftarrow N + \bar{i}$

3: **for** d **from** 0 **to** $(\log N - 1)$ **do**

4: $\quad path[d] \leftarrow nodes[(j \oplus 1) - 1]$

5: $\quad j \leftarrow j \gg 1$

6: **end for**

7: **return** $path[0:\log N]$

8.3.3 GGM 트리 재구성 함수 ReconstructTree

GGM 트리 재구성 함수 ReconstructTree(알고리즘 20)는 λ비트 길이의 비트열 $salt$, 비트열의 배열로 구성된 공개 경로 $path \in (\{0,1\}^\lambda)^{\log N}$, 정수 $k \in [\tau]$, 비공개 인덱스 $\bar{i} \in [N]$를 입력으로 받는다. 비공개 인덱스 $\bar{i}$로부터 재구성할 노드의 인덱스를 계산하고, 해당 노드의 값을 공개 경로의 값으로 설정한다. 해시 함수 H_4(**8.1.5** 및 알고리즘 11 참조)를 사용하여 설정된 노드 값으로부터 비공개 인덱스를 제외한 $N-1$개의 단말 노드를 재계산한다. 재계산된 $N-1$개의 단말 노드와 0^λ로 초기화된 공개하지 않는 단말 노드를 포함하여 $seeds$로 구성하여 출력한다.

ReconstructTree 함수는 검증 과정에서 사용한다.

알고리즘 20 ReconstructTree$(salt, path, k, \bar{i})$ — GGM 트리 재구성 함수

입력: 비트열 $salt \in \{0,1\}^\lambda$, 공개 경로 $path \in (\{0,1\}^\lambda)^{\log N}$, 정수 $k \in [\tau]$, 비공개 인덱스 $\bar{i} \in [N]$
출력: 재구성된 시드 $seeds \in (\{0,1\}^\lambda)^N$

1: $nodes[0:2N-1] \leftarrow \left(0^{\lambda}\right)^{2N-1}$
2: $j \leftarrow N + \bar{i}$
3: **for** d **from** 0 **to** $(\log N - 1)$ **do**
4: $\quad nodes[(j \oplus 1) - 1] \leftarrow path[d]$
5: $\quad j \leftarrow j \gg 1$
6: **end for**
7: **for** d **from** 1 **to** $(\log N - 1)$ **do**
8: $\quad$ **for** i **from** 2^d **to** $2^{d+1} - 1$ **do**
9: $\quad\quad j \leftarrow \left(\left(N + \bar{i}\right) \gg (\log N - d)\right)$
10: $\quad\quad$ **if** $i \neq j$ **then**
11: $\quad\quad\quad (nodes[2i-1], nodes[2i]) \leftarrow \mathrm{H}_4(salt, k, i, nodes[i-1])$ ▷ **8.1.5** 및 알고리즘 11 참조
12: $\quad\quad$ **end if**
13: $\quad$ **end for**
14: **end for**
15: $seeds[0:N] \leftarrow nodes[N-1:2N-1]$
16: **return** $seeds$

9 일반사항

9.1 외부 함수와 내부 함수

키 생성 과정, 서명 과정, 검증 과정은 외부 함수(external function)와 내부 함수(internal function)로 분리한다. 외부 함수에서 난수 생성 및 유효성 검사를 수행한 후 내부 함수를 호출하여야 하며, 내부 함수만 단독으로 사용하여서는 안 된다.

9.2 난수 생성

키 생성 과정과 서명 과정에서는 난수 비트열의 사용이 요구된다. 난수 비트열 생성은 이 표준의 범위 밖이며, 난수 비트열 생성 방법은 ISO/IEC 18031을 참조한다.

10 키 생성 과정

10.1 AIMer 키 생성 외부 함수 AIMer_keygen

AIMer 키 생성 외부 함수 AIMer_keygen(알고리즘 21)은 검증 키 pk와 서명 키 sk를 생성한다. 알고리즘 21의 1행부터 8행까지는 각각 λ비트 길이의 비트열 pt와 iv를 생성한 뒤 키 생성 내부 함수 AIMer_keygen_internal(**10.2** 및 알고리즘 22 참조)가 성공적으로 비트열 ct를 생성하면, 이를 이용해 검증 키 $pk \in (\{0,1\}^{\lambda})^2$와 서명 키 $sk \in (\{0,1\}^{\lambda})^3$를 생성한다.

알고리즘 21 AIMer_keygen() — AIMer 키 생성 외부 함수

출력: 검증 키 $pk \in \{0,1\}^{n+\lambda}$, 서명 키 $sk \in \{0,1\}^{2n+\lambda}$

$ct \leftarrow \perp$
while $ct = \perp$ **do**
  $pt \leftarrow_{\$} \{0,1\}^{n}$ ▷ 인가된 난수발생기 사용
  $iv \leftarrow_{\$} \{0,1\}^{\lambda}$ ▷ 인가된 난수발생기 사용
  if $pt = \text{NULL}$ **or** $iv = \text{NULL}$ **then**
    return $\perp$ ▷ 난수 생성 실패 시 오류 표시
  $ct \leftarrow \text{AIMer\_keygen\_internal}(pt, iv)$ ▷ **10.2** 및 알고리즘 22 참조
end while
$sk \leftarrow (pt, iv, ct)$
$pk \leftarrow (iv, ct)$
return (sk, pk)

10.2 AIMer 키 생성 내부 함수 AIMer_keygen_internal

AIMer 키 생성 내부 함수 AIMer_keygen_internal(알고리즘 22)은 n비트 길이의 비트열 pt와 λ비트 길이의 비트열 iv를 입력으로 받아, 검증 키 pk와 서명 키 sk를 생성한다. 키 생성 내부 함수는 다음과 같이 수행한다.

- pt와 iv를 입력으로 하여 일방향 함수 AIM3_NoZeroInpSB(**6.2** 및 알고리즘 3 참조)를 통해 ct를 계산한다. 이때 생성이 실패할 경우 ct 대신 ⊥을 출력한다.

알고리즘 22 $\text{AIMer\_keygen\_internal}(pt, iv)$ — AIMer 키 생성 내부 함수

입력: 비트열 $pt \in \{0,1\}^{n}$, 비트열 $iv \in \{0,1\}^{\lambda}$
출력: 비트열 $ct \in \{0,1\}^{n}$ 혹은 실패 기호 $\perp$

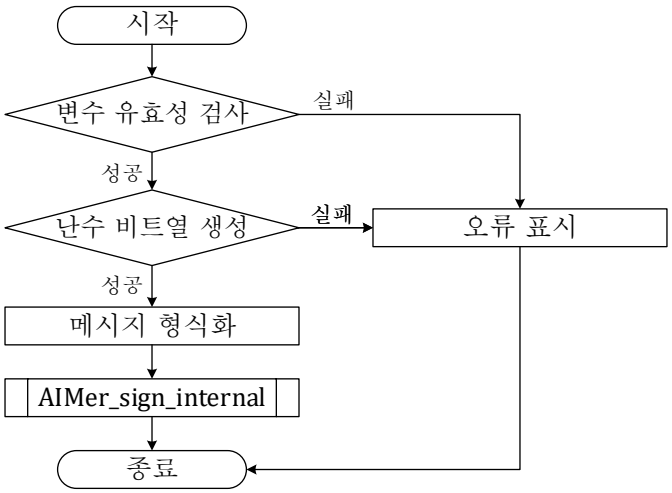
return AIM3_NoZeroInpSB(pt, iv) ▷ **6.2** 및 알고리즘 3 참조

11 서명 과정

11.1 일반사항

AIMer 서명 과정의 전체적인 동작을 도시하면 **그림 2**와 같다.

AIMer_sign:



AIMer_sign_internal:

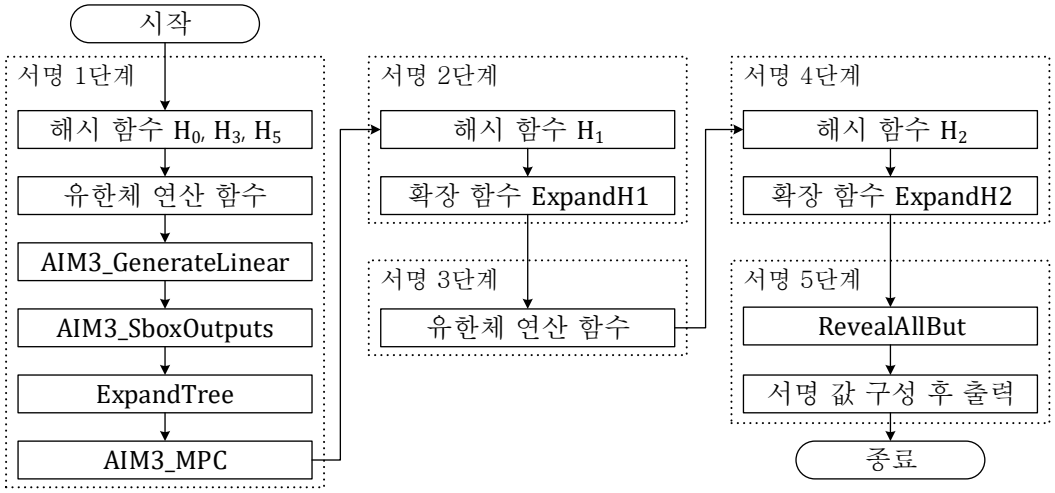


그림 2 — AIMer 서명 과정

11.2 AIMer 서명 외부 함수 AIMer_sign

AIMer 서명 외부 함수 AIMer_sign(알고리즘 23)은 서명 키 sk, 메시지 M, 컨텍스트 ctx를 입력으로 받는다. 알고리즘 23의 4행에서 난수 비트열을 생성할 때에는 **9.2**를 참조하여 λ비트 길이의 비트열 ρ를 생성하여야 한다. 결정론적 서명을 생성하려면 비트열 ρ를 0^{λ}으로 설정하여야 한다. 컨텍스트의 길이와 컨텍스트 및 메시지 M을 연접하여 형식화된 메시지 M'을 구성한 후, 서명 키 sk, 비트열 ρ과 함께 서명 내부 함수 AIMer_sign_internal(**11.2** 및 알고리즘 24 참조)에 입력하여 대응하는 서명 값 σ를 생성하고 비트열로 출력한다.

컨텍스트 ctx는 길이가 2 040 이하이며, 8의 배수인 비트열을 사용하여야 한다.

알고리즘 23 AIMer_sign(sk, M, ctx) — AIMer 서명 외부 함수

입력: 서명 키 $sk \in \{0,1\}^{2n+\lambda}$, 메시지 $M \in \{0,1\}^{*}$, 컨텍스트 ctx
출력: 서명 $\sigma \in (\{0,1\}^{\lambda})^{(5+(\log N+2\ell+5)\tau)}$

1: **If** $|ctx| > 2\,040$ **or** $|ctx|\ mod\ 8 \neq 0$ **then**

2: **return** $\perp$ ▷ 컨텍스트의 비트 길이가 2 040 초과이거나 8의 배수가 아닐 시 오류 표시

3: **end if**

4: $\rho \leftarrow_{\$} \{0,1\}^{\lambda}$ ▷ 결정론적 서명의 경우 $\rho \leftarrow 0^{\lambda}$

5: **if** $\rho = \mathrm{NULL}$ **then**

6: **return** $\perp$ ▷ 난수 생성 실패 시 오류 표시

7: **end if**

8: $M' \leftarrow \mathrm{IntToBits}((|ctx|/8),8) \parallel ctx \parallel M$ ▷ **5.1.1** 및 알고리즘 1 참조

9: **return** $\mathrm{AIMer\_sign\_internal}(sk, M', \rho)$ ▷ **11.2** 및 알고리즘 24 참조

11.3 AIMer 서명 내부 함수 AIMer_sign_internal

AIMer 서명 내부 함수 AIMer_sign_internal(알고리즘 24)은 서명 키 sk, 메시지 M', 비트열 ρ를 입력으로 받아, 대응하는 서명 값 σ를 생성하여 비트열로 출력한다. 서명 내부 함수는 아래와 같이 다섯 단계로 구성되어 있다.

- 서명 생성 1단계: 각 가상참가자들의 $seed$와 $seed$를 입력으로 한 해시 값 생성 및 $seed$에 대해 기록(commit)(**11.2.1** 참조)
- 서명 생성 2단계: 곱셈 검증 프로토콜(multiplication checking protocol)에 필요한 의사난수 비트열 만들기(**11.2.2** 참조)
- 서명 생성 3단계: 곱셈 검증 프로토콜의 출력 값 계산(**11.2.3** 참조)
- 서명 생성 4단계: 공개하지 않는 가상참여자 인덱스 계산하기(**11.2.4** 참조)
- 서명 생성 5단계: 공개 가상참여자에 대한 $seed$ 및 비공개 가상참여자의 기록 값 공개(**11.2.5** 참조)

알고리즘 24 $\mathrm{AIMer\_sign\_internal}(sk, M', \rho)$ — AIMer 서명 내부 함수

입력: 서명 키 $sk = (pt, iv, ct) \in \{0,1\}^{2n+\lambda}$, 메시지 $M' \in \{0,1\}^{*}$, 비트열 $\rho \in \{0,1\}^{\lambda}$
출력: 서명 $\sigma \in (\{0,1\}^{\lambda})^{(5+(\log N+2\ell+5)\tau)}$

1: $\mu \leftarrow \mathrm{H}_0(iv, ct, M')$ ▷ **8.1.1** 및 알고리즘 7 참조

2: $L \leftarrow \mathrm{AIM3\_GenerateLinear}(iv)$ ▷ **6.2.1** 및 알고리즘 4 참조

3: $(y_j)_{j\in[\ell+1]} \leftarrow \mathrm{AIM3\_SboxOutputs}(L, pt, ct)$ ▷ **6.3.1** 및 알고리즘 5 참조

4: $(salt, (seed_k)_{k\in[\tau]}) \leftarrow \mathrm{H}_3(\mu, pt, \rho)$ ▷ **8.1.4** 및 알고리즘 10 참조

5: **for** k **from** 0 **to** $(\tau - 1)$ **do**

6: $nodes_k[0: 2N-1] \leftarrow \mathrm{ExpandTree}(salt, k, seed_k)$ ▷ **8.3.1** 및 알고리즘 18 참조

7: $(seed_k^{(0)}, seed_k^{(1)}, \ldots, seed_k^{(N-1)}) \leftarrow nodes_k[N-1: 2N-1]$

8: **for** i **from** 0 **to** $(N-1)$ **do**

9: $\left(com_k^{(i)}, pt_k^{(i)}, (y_{k,j}^{(i)})_{j\in[\ell]}, (a_{k,j}^{(i)})_{j\in[\ell+1]}, c_k^{(i)}\right) \leftarrow \mathrm{H}_5(salt, k, i, seed_k^{(i)})$ ▷ **8.1.6** 및 알고리즘 12 참조

10: **end for**

11: $\Delta pt_k \leftarrow pt + \sum_{i\in[N]} pt_k^{(i)}$

12: $pt_k^{(N-1)} \leftarrow pt_k^{(N-1)} + \Delta pt_k$

13: **for** j **from** 0 **to** $(\ell - 1)$ **do**

14: $\Delta y_{k,j} \leftarrow y_j + \sum_{i\in[N]} y_{k,j}^{(i)}$

15: $y_{k,j}^{(N-1)} \leftarrow y_{k,j}^{(N-1)} + \Delta y_{k,j}$

16: **end for**

17: $\Delta c_k \leftarrow \sum_{j\in[\ell+1]}(Mul(\sum_{i\in[N]} a_{k,j}^{(i)}, y_j)) + \sum_{i\in[N]} c_k^{(i)}$ ▷ **8.2.1** 및 알고리즘 15 참조

18: $c_k^{(N-1)} \leftarrow c_k^{(N-1)} + \Delta c_k$

19: **for** i **from** 0 **to** $(N - 1)$ **do**

20: $\left(x_{k,j}^{(i)}, y_{k,j}^{(i)}, z_{k,j}^{(i)}\right)_{j\in[\ell+1]} \leftarrow \text{AIM3\_MPC}(L, ct, pt_k^{(i)}, \left(y_{k,j}^{(i)}\right)_{j\in[\ell]}, i)$ ▷ **6.3.2** 및 알고리즘 6 참조

21: **end for**

22: **end for**

23: $h_1 \leftarrow \text{H}_1(\mu, salt, ((com_k^{(i)})_{i\in[N]}, \Delta pt_k, (\Delta y_{k,j})_{j\in[\ell]}, \Delta c_k)_{k\in[\tau]})$ ▷ **8.1.2** 및 알고리즘 8 참조

24: $\left((\epsilon_{k,j})_{j\in[\ell+1]}\right)_{k\in[\tau]} \leftarrow \text{ExpandH1}(h_1)$ ▷ **8.1.7** 및 알고리즘 13 참조

25: **for** k **from** 0 **to** $(\tau - 1)$ **do**

26: **for** i **from** 0 **to** $(N - 1)$ **do**

27: **for** j **from** 0 **to** ℓ **do**

28: $\alpha_{k,j}^{(i)} \leftarrow a_{k,j}^{(i)} + \text{Mul}(x_{k,j}^{(i)}, \epsilon_{k,j})$ ▷ **8.2.1** 및 알고리즘 15 참조

29: **end for**

30: **end for**

31: **for** j **from** 0 **to** ℓ **do**

32: $\alpha_{k,j} \leftarrow \sum_{i\in[N]} \alpha_{k,j}^{(i)}$

33: **end for**

34: **for** i **from** 0 **to** $(N - 1)$ **do**

35: $v_k^{(i)} \leftarrow c_k^{(i)} + \sum_{j\in[\ell+1]}\left(\text{Mul}\left(z_{k,j}^{(i)}, \epsilon_{k,j}\right) + \text{Mul}(\alpha_{k,j}, y_{k,j}^{(i)})\right)$ ▷ **8.2.1** 및 알고리즘 15 참조

36: **end for**

37: **end for**

38: $h_2 \leftarrow \text{H}_2(h_1, salt, ((\alpha_{k,0}^{(i)}, \dots, \alpha_{k,\ell}^{(i)})_{i\in[N]}, (v_k^{(i)})_{i\in[N]})_{k\in[\tau]})$ ▷ **8.1.3** 및 알고리즘 9 참조

39: $\left(\bar{i}_k\right)_{k\in[\tau]} \leftarrow \text{ExpandH2}(h_2)$ ▷ **8.1.8** 및 알고리즘 14 참조

40: **for** k **from** 0 **to** $(\tau - 1)$ **do**

41: $path_k \leftarrow \text{RevealAllBut}(nodes_k, \bar{i}_k)$ ▷ **8.3.2** 및 알고리즘 19 참조

42: **end for**

43: $\sigma \leftarrow \left(salt, h_1, h_2, \left(path_k, com_k^{(\bar{i}_k)}, \Delta pt_k, \left(\Delta y_{k,j}\right)_{j\in[\ell]}, \Delta c_k, \left(\alpha_{k,j}^{(\bar{i}_k)}\right)_{j\in[\ell+1]}\right)_{k\in[\tau]}\right)$

44: **return** σ

11.3.1 서명 생성 1단계

a) 메시지 M'에 대한 해시 값 μ를 구한다(1행).
b) pt와 iv를 이용하여 AIM3의 S-box 계산 값과 선형층을 계산한다(2-3행).
c) 일회용 난수 ρ를 사용하여 $salt$와 GGM 트리의 루트 시드 값들을 생성한다(4행).
d) 아래 과정을 τ번 반복한다(5-22행).
   1) 루트 시드로부터 단말 노드의 개수가 가상참가자 수 N과 같은 GGM 트리를 생성하여 N개의 단말 노드를 각 가상참가자의 $seed$로 사용한다(6-7행).
   2) 각 가상참가자의 $seed$를 기록하고 $seed$를 입력으로 한 해시 값 random tape를 생성한다(8-9행).
   3) 생성된 random tape 중에서 $pt_k^{(i)}$의 합이 pt와 같게 되도록 보정한다(11-12행).
   4) 생성된 random tape 중에서 $\left(y_{k,j}^{(i)}\right)_{j\in[\ell]}$의 합이 $\left(y_j\right)_{j\in[\ell]}$와 같게 되도록 보정한다(13-16행).
   5) 생성된 random tape 중에서 곱셈 검증 프로토콜(**11.2.3** 참조)에서 사용되는 $\left(a_{k,j}^{(i)}\right)_{j\in[\ell+1]}, c_k^{(i)}$를 $\sum_{j\in[\ell+1]}\left(\sum_i a_{k,j}^{(i)}\right)\cdot y_j = \left(\sum_i c_k^{(i)}\right)$ 식을 항상 만족하도록 보정한다(17-18행).
   6) AIM3_MPC(**6.3.2** 참조) 함수를 수행하여 곱셈 검증 프로토콜(**11.2.3** 참조)에서 사용되는 $\left(x_{k,j}^{(i)}, y_{k,j}^{(i)}, z_{k,j}^{(i)}\right)_{j\in[\ell+1]}$를 계산한다(19-21행).

11.3.2 서명 생성 2단계

해시 함수 H_1을 사용하여 해시 값 h_1을 계산하고, 확장 함수 ExpandH1에 h_1를 입력하여 곱셈 검증 프로토콜에서 사용하는 $\left((\epsilon_{k,j})_{j\in[\ell+1]}\right)_{k\in[\tau]}$ 값을 계산한다(23-24행).

11.3.3 서명 생성 3단계

곱셈 검증 프로토콜의 출력 값 $\left(\left(\alpha_{k,0}^{(i)},\ldots,\alpha_{k,\ell}^{(i)}\right)_{i\in[N]}\right)_{k\in[\tau]}$ 과 $\left(\left(v_k^{(i)}\right)_{i\in[N]}\right)_{k\in[\tau]}$를 구한다(25-37행).

11.3.4 서명 생성 4단계

해시 값 h_2를 계산하고, 확장 함수 ExpandH2에 h_2를 입력하여 공개하지 않는 가상참가자 인덱스 $\left(\bar{i}_k\right)_{k\in[\tau]}$를 계산한다(38-39행).

11.3.5 서명 생성 5단계

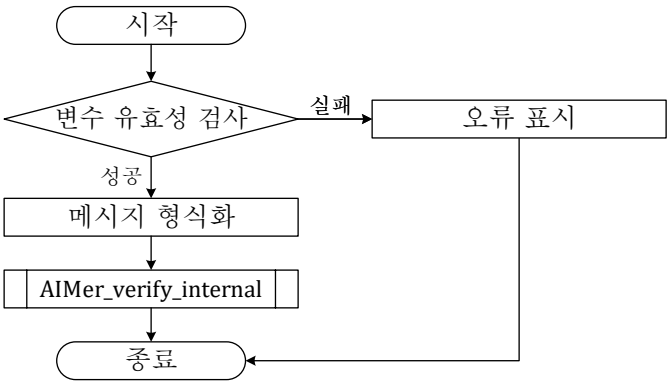
공개하지 않는 가상참가자를 제외한 $N-1$ 가상참가자들의 $seed$ 값을 공개하기 위해 GGM 트리의 공개 경로 $(path_k)_{k\in[\tau]}$를 생성하고, 서명 값 σ를 구성하여 출력한다(40-44행).

12 검증 과정

12.1 일반사항

AIMer 검증 과정의 전체적인 동작을 도시하면 **그림 3**과 같다.

AIMer_verify:



AIMer_verify_internal:

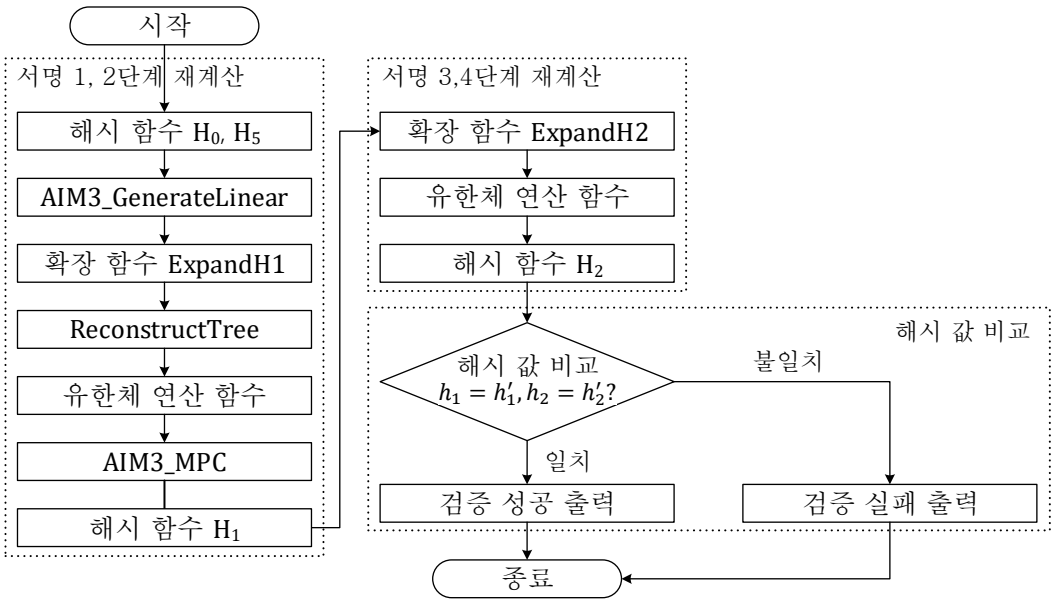


그림 3 — AIMer 검증 과정

12.2 AIMer 검증 외부 함수 AIMer_verify

AIMer 검증 외부 함수 AIMer_verify(알고리즘 25)는 검증 키 pk, 메시지 M, 검증할 서명 σ, 컨텍스트 ctx를 입력으로 받는다. 컨텍스트의 길이와 컨텍스트, 메시지 M을 연접하여 형식화된 메시지 M'을 구성한 후, 검증 키 pk, 서명 값 σ와 함께 검증 내부 함수 AIMer_verify_internal(**12.2** 및 알고리즘 26 참조)에 입력하여, 검증에 성공하면 $true$, 실패하면 $false$를 출력한다.

컨텍스트 ctx는 길이가 2 040 이하이며, 8의 배수인 비트열을 사용하여야 한다.

알고리즘 25 AIMer_verify(pk, M, σ, ctx) — AIMer 검증 외부 함수

입력: 검증 키 $pk \in \{0,1\}^{n+\lambda}$, 메시지 $M \in \{0,1\}^*$, 서명 $\sigma \in (\{0,1\}^{\lambda})^{(5+(\log N+2\ell+5)\tau)}$, 컨텍스트 ctx
출력: 검증 성공 $true$ 또는 검증 실패 $false$

1: **If** $|ctx| > 2\,040$ **or** $|ctx| \bmod 8 \neq 0$ **then**
2: **return** $\perp$ ▷ 컨텍스트의 비트 길이가 2 040 초과이거나 8의 배수가 아닐 시 오류 표시

3: **end if**

4: $M' \leftarrow \mathrm{IntToBits}((|ctx|/8),8) \,\|\, ctx \,\|\, M$ ▷ **5.1.1** 및 알고리즘 1 참조

5: **return** AIMer_verify_internal(pk, M', σ) ▷ **12.2** 및 알고리즘 26 참조

12.3 AIMer 검증 내부 함수 AIMer_verify_internal

AIMer 검증 내부 함수 AIMer_verify_internal(알고리즘 26)은 검증 키 $pk = (iv, ct)$, 메시지 M', 서명 σ을 입력으로 받아 검증 성공 $true$ 혹은 검증 실패 $false$를 출력한다. 검증 내부 함수의 수행 절차는 다음과 같다.

- 검증 키 $pk = (iv, ct)$를 이용하여 AIM3의 선형층을 계산하고, 메시지 M'을 이용하여 해시 값 μ를 계산
- 입력 받은 서명 값에 포함된 해시 값 h_1, h_2를 이용하여 서명 생성 과정의 2, 4단계에서 계산하는 확장 함수 값을 재계산
- 서명 생성 1, 2단계 재계산(**12.2.1** 참조)
- 서명 생성 3, 4단계 재계산(**12.2.2** 참조)
- 해시 값 비교(**12.2.3** 참조)

알고리즘 26 AIMer_verify_internal(pk, M', σ) — AIMer 검증 내부 함수

입력: 검증 키 $pk = (iv, ct) \in \{0,1\}^{n+\lambda}$, 메시지 $M' \in \{0,1\}^*$, 서명 σ
출력: 검증 성공 $true$ 또는 실패 $false$

$\left(salt, h_1, h_2, \left(path_k, com_k^{(\bar{i}_k)}, \Delta pt_k, (\Delta y_{k,j})_{j\in[\ell]}, \Delta c_k, \left(a_{k,j}^{(\bar{i}_k)}\right)_{j\in[\ell+1]}\right)_{k\in[\tau]}\right) \leftarrow \sigma$

$\mu \leftarrow \mathrm{H}_0(iv, ct, M')$ ▷ **8.1.1** 및 알고리즘 7 참조

$L \leftarrow \mathrm{AIM3\_GenerateLinear}(iv)$ ▷ **6.2.1** 및 알고리즘 4 참조

$\left((\epsilon_{k,j})_{j\in[\ell+1]}\right)_{k\in[\tau]} \leftarrow \mathrm{ExpandH1}(h_1)$ ▷ **8.1.7** 및 알고리즘 13 참조

$\left(\bar{i}_k\right)_{k\in[\tau]} \leftarrow \mathrm{ExpandH2}(h_2)$ ▷ **8.1.8** 및 알고리즘 14 참조

for k **from** 0 **to** $(\tau - 1)$ **do**

  $\left(seed_k^{(0)}, seed_k^{(1)}, \ldots, seed_k^{(N-1)}\right) \leftarrow \mathrm{ReconstructTree}(salt, path_k, k, \bar{i}_k)$ ▷ **8.3.3** 및 알고리즘 20 참조

  for i **from** 0 **to** $(N-1)$, $i \neq \bar{i}_k$ **do**

    $\left(com_k^{(i)}, pt_k^{(i)}, \left(y_{k,j}^{(i)}\right)_{j\in[\ell]}, \left(a_{k,j}^{(i)}\right)_{j\in[\ell+1]}, c_k^{(i)}\right) \leftarrow \mathrm{H}_5(salt, k, i, seed_k^{(i)})$ ▷ **8.1.6** 및 알고리즘 12 참조

    if $i = N - 1$ **then**

      $pt_k^{(N-1)} \leftarrow pt_k^{(N-1)} + \Delta pt_k$

      for j **from** 0 **to** $(\ell - 1)$ **do**

        $y_{k,j}^{(N-1)} \leftarrow y_{k,j}^{(N-1)} + \Delta y_{k,j}$

      end for

      $c_k^{(N-1)} \leftarrow c_k^{(N-1)} + \Delta c_k$

    end if

    $\left(x_{k,j}^{(i)}, y_{k,j}^{(i)}, z_{k,j}^{(i)}\right)_{j\in[\ell+1]} \leftarrow \mathrm{AIM3\_MPC}(L, ct, pt_k^{(i)}, \left(y_{k,j}^{(i)}\right)_{j\in[\ell]}, i)$ ▷ **6.3.2** 및 알고리즘 6 참조

18: **for** j **from** 0 **to** ℓ **do**

19: $\alpha_{k,j}^{(i)} \leftarrow a_{k,j}^{(i)} + \mathrm{Mul}(x_{k,j}^{(i)}, \epsilon_{k,j})$ ▷ **8.2.1** 및 알고리즘 15 참조

20: **end for**

21: **for** j **from** 0 **to** ℓ **do**

22: $\alpha_{k,j} \leftarrow \sum_{i\in[N]} \alpha_{k,j}^{(i)}$

23: **end for**

24: $v_k^{(i)} \leftarrow c_k^{(i)} + \sum_{j\in[\ell+1]}(\mathrm{Mul}(z_{k,j}^{(i)}, \epsilon_{k,j}) + \mathrm{Mul}(\alpha_{k,}y_{k,j}^{(i)}))$ ▷ **8.2.1** 및 알고리즘 15 참조

25: $v_k^{(\bar{i}_k)} \leftarrow \sum_{i\in[N]\setminus\{\bar{i}_k\}} v_k^{(i)}$

26: **end for**

27: **end for**

30: $h_1' \leftarrow \mathrm{H}_1(\mu, salt, ((com_k^{(i)})_{i\in[N]}, \Delta pt_k, (\Delta y_{k,j})_{j\in[\ell]}, \Delta c_k)_{k\in[\tau]})$ ▷ **8.1.2** 및 알고리즘 8 참조

31: $h_2' \leftarrow \mathrm{H}_2(h_1', salt, ((\alpha_{k,0}^{(i)}, \dots, \alpha_{k,\ell}^{(i)})_{i\in[N]}, (v_k^{(i)})_{i\in[N]})_{k\in[\tau]})$ ▷ **8.1.3** 및 알고리즘 9 참조

32: **if** $h_1 = h_1'$ **and** $h_2 = h_2'$ **then**

33: **return** $true$

34: **else**

35: **return** $false$

36: **end if**

12.3.1 서명 생성 1, 2단계 재계산

아래의 과정을 τ번 반복한다.

- 입력으로 받은 서명 값에 포함된 공개 경로 $path_k$를 이용하여 공개된 가상참가자들의 $seed$를 재계산하고, 기록과 random tape를 재생성
- 재생성된 기록과 random tape, 보정 값을 이용하여, 해시 값 h_1'을 재계산

12.3.2 서명 생성 3, 4단계 재계산

아래의 과정을 τ번 반복한다.

- 공개된 가상참가자들에 대하여 곱셈 검증 프로토콜을 시뮬레이션한다.
- 공개된 가상참가자들의 곱셈 검증 프로토콜 출력 값을 계산하여, 공개되지 않은 가상참가자의 곱셈 검증 프로토콜 출력 값 $v_k^{(\bar{i}_k)}$를 재계산
- 해시 값 h_2'을 재계산

12.3.3 해시 값 비교

12.2.1과 **12.2.2**에서 재계산된 해시 값 h_1', h_2'과 서명 값에 포함된 해시 값 h_1, h_2를 비교하여, h_1과 h_1', h_2와 h_2'가 모두 일치하면 검증 성공 $true$를 출력하고, 하나라도 일치하지 않으면 검증 실패 $false$를 출력한다.

참고문헌

1. J. Lee, J. Cho, J. Ha, J. Kwon, S. Kim, B. Lee, J. Lee, S. Lee, D. Moon, M. Son, and H. Yoon. "The AIMer Signature Scheme (Ver 3.0)". 2026. 배포처: AIMer team 홈페이지(https://aimer-signature.org)
2. Oded Goldreich, Shafi Goldwasser, and Silvio Micali. 1986. “How to construct random functions”. J. ACM 33, 4 (Oct. 1986), 792–807. https://doi.org/10.1145/6490.6503