코드 기반 전자서명 CROSS

https://youtu.be/6IkflTur5Rk

NIST PQC 추가 전자서명 알고리즘 공모전

- 2022년 9월 NIST는 양자 내성 전자서명의 기반 문제를 다양화 하기 위해서 PQC 표준화 공모전을 소개함
- 2023년 6월 1라운드에 50개의 서명 제출
- **2023년 7월 1라운드 후보군 알고리즘으로 40개의 알고리즘 선정**
  - Code-based 6개 Isogeny 1개, Latticed-base 7개, MPC-in-the-Head 7개, Multivariate 10개, Symmetric-based 4개, other-based 5개

MPC-in-the-Head :Multi-Party Computation-in-the-Head(MPCitH)

- **2024년 10월 2라운드 후보군 알고리즘으로 14개의 알고리즘 선정**
  - KpqC 알고리즘 표준화 선정 알고리즘인 HAETAE와 AIMer는 1라운드 후보군이었지만 2라운드에는 선정되지 못하였음

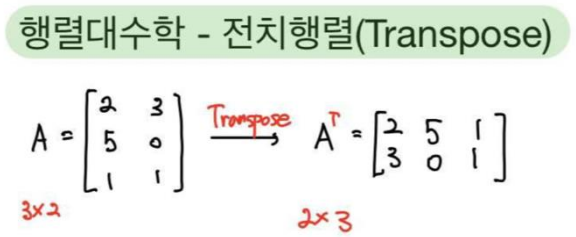
Code-based	Isogeny	Lattice-based	MPC-in-the-Head	Multivariate	Symmetric-based
CROSS	SQIsign	HAWK	Mirath	MAYO	FAEST
LESS			MQOM	QR-UOV	
			PERK	SNOVA	
			RYDE	UOV	
			SDitH		

https://csrc.nist.gov/projects/pqc-dig-sig/round-2-additional-signatures

Code-based

- Code-based의 Code는 **Error-Correcting Code**(오류 정정 부호)를 의미
- 오류 정정 부호는 통신 채널에서 발생하는 노이즈(오류)를 복구하기 위해 사용됨
  - Linear code(선형 부호) :
  (송신) 메시지 벡터 x **생성행렬(Generator Matrix)** = Codeword 생성
  (수신) **검증 행렬(Parity-Check Matrix)**등으로 오류를 검출 및 정정
- 코드 기반 암호의 특징
  - ”특정 부호(코드)의 구조를 역으로 추론해서 메시지를 복구하기가 어려운 문제”를 사용
  - 코드워드에 인위적으로 노이즈(오류)를 추가한 상태에서 원본 메시지를 복원하려면 **부호 구조**(오류 정정 능력)과 “**Private Key**”를 알아야함
- 코드기반 암호는 선형 부호의 오류 정정 문제(특히 신드롬 디코딩)의 난해성을 근간으로 함

Code-based 용어 정리



- **Syndrome Decoding**
  - 패리티 체크 행렬 $\boldsymbol{H} \in \mathbb{F}_2^{r \times n}$ 과 신드롬 $\mathbf{s}$ 가 주어졌을 때 $\boldsymbol{H} \cdot \boldsymbol{e}^{\boldsymbol{T}} = \boldsymbol{s}$
    를 만족하는 에러 벡터 $\mathbf{e}$를 찾는 문제를 의미(T는 Transpose;전치연산을 의미/ e를 가로로 펴 놓은 행벡터
- **QC(Quasi-Cyclic) 부호**
  - QC 부호는 한 블록(길이 p)을 기준으로 하는 순환 (cyclic) 구조를 갖는 행렬을 블록 단위로 배치한 형태

  Ex) 패리티 체크 행렬 $\boldsymbol{H}$ 가 $r \times r$개의 블록으로 구성되어 있고, 각 블록은 $p \times p$ 크기의 순환 행렬
  - 순환 행렬은 한 행이 바로 윗 행을 한 칸 오른쪽으로 shift한 형태로 구성되는 특별한 구조
  - $\mathbb{F}_2[x]/(x^p - 1)$**와 같은 다항식 링 표현을 자주 사용**
  - 곱셈 연산 등이 곧 순환 행렬과의 곱셈에 대응되어 효율적인 덧셈과 곱셈이 가능
- **LDPC(Low-Density Parity-Check)**
  - 패리티 체크 행렬 $\boldsymbol{H}$가 매우 희소(sparse)한 부호이며, 일반적으로 열당, 행당 적은 수의 1이 존재함
- **MDPC(Moderate-Density Parity-Check) (CROSS는 여기에 해당)**
  - LDPC보다 '조금 더 높은 밀도'(1의 개수가 약간 더 많은 행렬을 의미)를 갖는 부호
  - 반복 디코딩 알고리즘(bit filping등)으로 에러를 정정
  - Goppa 기반보다 구현이 단순하고 빠름

Code-based 용어 정리

- **스크램블 행렬 S**
  - 패리티 체크 행렬 H나 Generator 행렬 G 에 왼쪽에서 곱하여 해밍웨이트 등을 뒤섞어버리는 가역 행렬
  - 구조적 공격자가 “어떤 QC 부호를 쓰는지” 추론하기 어렵게 함
- **McEliece 형태의 프레임워크**
  - G가 제너레이터 행렬이 될 수 있도록 특정한 선형 부호 준비
  - 이를 은폐하기 위해 무작위 행렬(permutation 행렬 등)을 곱해서 공개키 생성
  - 제너레이터 행렬 관점 : $c = mG + e$
- **Niederreiter 형태의 프레임워크(CROSS는 여기에 해당)**
  - 패리티 체크 행렬을 활용하고 에러 벡터를 이용해 암호문 생성
  - 패리티체크 행렬 관점 : $c = H \cdot e^T$

Code-based 용어 정리

- **Syndrome Decoding Problem(SDP)**
  - H를 이용해 오류벡터 e를 복원(디코딩)하는 문제
  - H가 $(n-k) \times n$ 크기의 행렬이고, 주어진 신드롬 s 벡터가 $e \cdot H^T = s$
  를 만족하는 e를 찾는 것
- **Restricted- SDP (CROSS는 여기에 해당)**
  - 에러 벡터 e가 특정한 군(subgroup)에 속한다 등의 '제한'을 갖음

  Ex) e의 각 좌표가 F_p 안에서 특정 부분군 E에 속한다
  - 이렇게 하면 공격자가 e를 찾기 어렵게 하는 효과
  - 동시 서명할 때 e나 transform에 대한 정보를 짧게 표현 가능
- **Restricted- SDP(G) (CROSS는 여기에 해당)**
  - R-SDP의 확장으로, 에러 벡터 전체가 특정 부분군에 속해야함
  - R-SDP보다 더 빠른 서명이 가능하지만, 구조가 복잡하여 구현은 복잡함

CROSS

- **Restricted Syndrome Decoding Problem(R-SDP)**를 기반으로 함
- Fiat-Shamir 변환을 사용해 interactive(대화형) 영지식 증명 프로토콜을 비대화형 서명 체계로 변경
- **Quasi-Cyclic(QC) 구조를 통해 공개키 크기를 크게 줄임**
  - 코드 기반 암호들은 공개키 크기가 크다는 단점 해결
- **스크램블링(Scrambling)+Permutation(대치)행렬**로 원본 코드 구조 노출되는 공격에 대응
- **MDPC(Moderate-Density Parity-Check)** 계열의 디코딩 알고리즘 적용
  - 상대적으로 빠른 연산과 낮은 Decoding 실패율 추구
- 낮은 연산 복잡도를 통해 서명 및 검증 시 빠른 처리 속도 달성
- $\mathbb{F}_2[x]/(x^p - 1)$기반 다항식 연산으로 인해 HW/SW 병렬화에 유리
  - P는 QC 구조에서의 순환 블록 크기(부호의 크기)를 의미

CROSS

- p: modular arithmetic
- z : 부분 치환(sub-block) 단위
- n: 전체 블록의 dimension
- k : 랜덤 계수의 개수
- m : 처리할 메세지 블록의 수
- t : 치환 연산의 반복 횟수(라운드 수)
- w : 추가 노이즈

Table 4: Parameter choices, keypair and signature sizes recommended for both CROSS-R-SDP and CROSS-R-SDP(G), assuming NIST categories 1, 3, and 5, respectively.

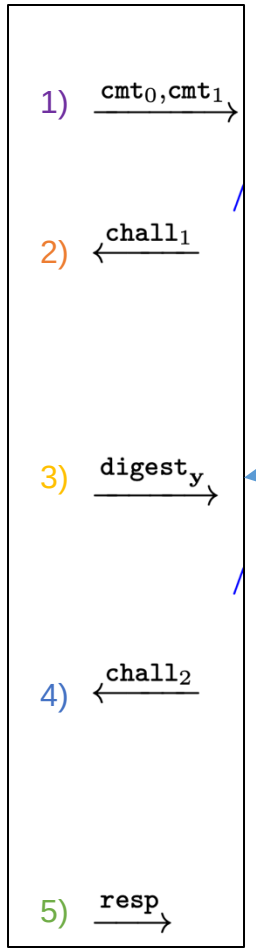
Algorithm and Security Category	Optim. Corner	p	z	n	k	m	t	w	Pri. Key Size (B)	Pub. Key Size (B)	Signature Size (B)
CROSS-R-SDP **1**	**fast**	127	7	127	76	-	157	82	32	77	18432
	balanced	127	7	127	76	-	256	215	32	77	13152
	small	127	7	127	76	-	520	488	32	77	12432
CROSS-R-SDP **3**	**fast**	127	7	187	111	-	239	125	48	115	41406
	balanced	127	7	187	111	-	384	321	48	115	29853
	small	127	7	187	111	-	580	527	48	115	28391
CROSS-R-SDP **5**	**fast**	127	7	251	150	-	321	167	64	153	74590
	balanced	127	7	251	150	-	512	427	64	153	53527
	small	127	7	251	150	-	832	762	64	153	50818
CROSS-R-SDP(G) **1**	**fast**	509	127	55	36	25	147	76	32	54	11980
	balanced	509	127	55	36	25	256	220	32	54	9120
	small	509	127	55	36	25	512	484	32	54	8960
CROSS-R-SDP(G) **3**	**fast**	509	127	79	48	40	224	119	48	83	26772
	balanced	509	127	79	48	40	268	196	48	83	22464
	small	509	127	79	48	40	512	463	48	83	20452
CROSS-R-SDP(G) **5**	**fast**	509	127	106	69	48	300	153	64	106	48102
	balanced	509	127	106	69	48	356	258	64	106	40100
	small	509	127	106	69	48	642	575	64	106	36454

CROSS-ID

- Interactive(대화형) 영지식 증명 프로토콜
- '내가 특정 R-SDP(G)의 해답 e 를 알고 있다' 는
  사실을 영지식으로 증명할 수 있게 함
  - 5번의 메시지를 주고 받는 방식

5-pass protocol

1. 증명자가 임의 값으로 커밋(commit)
2. 검증자가 챌린지(challenge)1을 보냄
3. 증명자가 응답(response)1
4. 검증자가 챌린지 2를 보냄
5. 증명자가 응답 2



Private Key $\mathbf{e} \in G$
Public Key $G \subseteq \mathbb{E}^n$, $\mathbf{H} \in \mathbb{F}_p^{(n-k)\times n}$, $\mathbf{s} = \mathbf{e}\mathbf{H}^\top \in \mathbb{F}_p^{n-k}$

PROVER		VERIFIER
// Sampling Seed to compute $\mathbf{e}', \mathbf{u}'$		
$\mathtt{Seed} \xleftarrow{\$} \{0,1\}^\lambda$		
$(\mathbf{e}', \mathbf{u}')$	$\leftarrow$ CSPRNG(Seed)	
// with co − domain $G \times \mathbb{F}_p^n$		
// Computing $\mathbf{v}, \mathbf{u}, \mathbf{s}'$		
$\mathbf{v} \leftarrow \mathbf{e} \star (\mathbf{e}')^{-1}$		
$\mathbf{u} \leftarrow \mathbf{v} \star \mathbf{u}'$		
$\mathbf{s}' \leftarrow \mathbf{u}\mathbf{H}^\top$		
// Computing commitments		
$\mathtt{cmt}_0 \leftarrow \mathsf{Hash}(\mathbf{s}' \mid \mathbf{v})$		
$\mathtt{cmt}_1 \leftarrow \mathsf{Hash}(\mathbf{u}' \mid \mathbf{e}')$		
	$\mathtt{cmt}_0, \mathtt{cmt}_1$ →	
	// Sampling first challenge	
	$\leftarrow \mathtt{chall}_1$	$\mathtt{chall}_1 \xleftarrow{\$} \mathbb{F}_p^*$
// Computing first response		
$\mathbf{y} \leftarrow \mathbf{u}' + \mathtt{chall}_1 \mathbf{e}'$		
$\mathtt{digest}_\mathbf{y} \leftarrow \mathsf{Hash}(\mathbf{y})$		
	$\mathtt{digest}_\mathbf{y}$ →	
	// Sampling second challenge	
	$\leftarrow \mathtt{chall}_2$	$\mathtt{chall}_2 \xleftarrow{\$} \{0,1\}$
// Computing second response		
If $\mathtt{chall}_2 = 0$, $\mathtt{resp} \leftarrow (\mathbf{y}, \mathbf{v})$		
If $\mathtt{chall}_2 = 1$, $\mathtt{resp} \leftarrow \mathtt{Seed}$		
	$\mathtt{resp}$ →	
	// Verification	
	If $\mathtt{chall}_2 = 0$:	
	$\mathbf{y}' \leftarrow \mathbf{v} \star \mathbf{y}$	
	$\mathbf{s}' \leftarrow \mathbf{y}'\mathbf{H}^\top - \mathtt{chall}_1 \mathbf{s}$	
	Accept if:	
	1) $\mathsf{Hash}(\mathbf{y}) = \mathtt{digest}_\mathbf{y}$	
	2) $\mathsf{Hash}(\mathbf{s}' \mid \mathbf{v}) = \mathtt{cmt}_0$	
	3) $\mathbf{v} \in G$	
	If $\mathtt{chall}_2 = 1$:	
	$(\mathbf{e}', \mathbf{u}') \leftarrow \mathrm{CSPRNG}(\mathtt{Seed})$ // with co − domain $G \times \mathbb{F}_p^n$	
	$\mathbf{y} \leftarrow \mathbf{u}' + \mathtt{chall}_1 \mathbf{e}'$	
	Accept if:	
	1) $\mathsf{Hash}(\mathbf{y}) = \mathtt{digest}_\mathbf{y}$	
	2) $\mathsf{Hash}(\mathbf{u}' \mid \mathbf{e}') = \mathtt{cmt}_1$	

Figure 2: CROSS-ID

CROSS-ID

- 5-pass 구조를 실제 서비스에서 사용하는 것은 비효율적
  - 매번 검증자가 질의를 만들어야하는 “대화형”이기 때문
- 따라서, **Fiat-Shamir** 사용
  - 해시 함수를 이용해 챌린지를 스스로 생성하도록 바꾸는 것
  - 프로토콜을 **비대화형(non-interactive)** 증명으로 만들 수 있고, 이를 **전자서명 알고리즘**으로 확장 가능함.
- **T번 반복 병렬 실행 + Fiat-Shamir → 서명 (한꺼번에 t라운드)**

<전자서명 알고리즘으로 확장 가능한 이유?>

- 대화형 증명은 검증자가 보낸 챌린지가 필요했음
- **Fiat-Shamir 방식은 해시로 만든 챌린지가 되기 때문에 “메시지+커밋+챌린지+응답”을 하나의 패키지로 만들 수 있음**
  - Fiat-Shamir 방식은 검증자에게 랜덤 챌린지를 받는 단계를 해시 함수로 챌린지를 내부에서 생성하는 것으로 바꿔서 대화 과정을 제거함
- **최종 커밋-챌린지-응답을 하나의 서명(트랜스크립트)으로 묶어 공개하면 검증자는** 메시지와 공개된 트랜스크립트를 활용해 **똑같이 챌린지를 재생성해볼 수 있음**

CROSS 키 생성

- Seed_sk를 임의로 뽑음
- 이 시드로부터 난수 생성기를 돌려서
  패리티 체크 행렬 H와 비밀 벡터 e 등을 만듦
- $\boldsymbol{e} \cdot \boldsymbol{H}^{\boldsymbol{T}} = \boldsymbol{s}$ 를 계산해서 신드롬 s 구함
- 공개키 = (Seed_pk, s) 형태
- 비밀키 = Seed_sk

Algorithm 1: KeyGen()

Input: None
Output: $\mathtt{sk} : \mathtt{Seed}_{\mathtt{sk}}$: secret key seed;
$\mathtt{pk} : (\mathtt{Seed}_{\mathtt{pk}}, \mathbf{s})$ public key;
Data: λ: security parameter;
$g \in \mathbb{F}_p^*$: generator of $\mathbb{E}$;

// Sampling seeds
$\mathtt{Seed}_{\mathtt{sk}} \xleftarrow{\$} \{0,1\}^{2\lambda}$
$(\mathtt{Seed}_{\mathbf{e}}, \mathtt{Seed}_{\mathtt{pk}}) \leftarrow \mathsf{CSPRNG}_{-\{0,1\}^{2\lambda} \times \{0,1\}^{2\lambda}} (\mathtt{Seed}_{\mathtt{sk}} \mid 3t+1)$

// Sampling random matrices $\mathbf{H}$ and $\overline{\mathbf{M}}$
$(\overline{\mathbf{W}}, \mathbf{V}) \leftarrow \mathsf{CSPRNG}_{-\mathbb{F}_z^{m \times (n-m)} \times \mathbb{F}_p^{(n-k) \times k}} (\mathtt{Seed}_{\mathtt{pk}} \mid 3t+2)$ | $\mathbf{V} \leftarrow \mathsf{CSPRNG}_{-\mathbb{F}_p^{(n-k) \times k}} (\mathtt{Seed}_{\mathtt{pk}} \mid 3t+2)$
$\mathbf{H} \leftarrow [\mathbf{V} \mid \mathrm{Id}_{n-k}]$

// Computing $\mathbf{e}$
$\overline{\mathbf{M}} \leftarrow [\overline{\mathbf{W}} \mid \mathrm{Id}_m]$
$\overline{\mathbf{e}}_G \leftarrow \mathsf{CSPRNG}_{-\mathbb{F}_z^m} (\mathtt{Seed}_{\mathbf{e}} \mid 3t+3)$ | $\overline{\mathbf{e}} \leftarrow \mathsf{CSPRNG}_{-\mathbb{F}_z^n} (\mathtt{Seed}_{\mathbf{e}} \mid 3t+3)$
$\overline{\mathbf{e}} \leftarrow \overline{\mathbf{e}}_G \overline{\mathbf{M}}$
for j **from** 1 **to** n **do**
  $\mathbf{e}_j \leftarrow g^{\overline{\mathbf{e}}_j}$

// Computing the syndrome s
$\mathbf{s} \leftarrow \mathbf{e}\mathbf{H}^\top$

// Return secret key sk and public key pk
$\mathtt{sk} \leftarrow \mathtt{Seed}_{\mathtt{sk}}$
$\mathtt{pk} \leftarrow (\mathtt{Seed}_{\mathtt{pk}}, \mathbf{s})$
return $(\mathtt{sk}, \mathtt{pk})$;

CROSS 서명

- 서명하고자 하는 메시지와 함께 여러 라운드(**t회**) 영지식 증명 구조를 한 번에 구성.
- 각 라운드마다 임의 벡터(u′, e′)를 뽑아서 commit을 만들고, 해시로부터 chall1, chall2(두 종류의 챌린지)를 뽑는다.
- chall2의 비트(weight w에 맞추어) 0이면 "(y[i],v[i]) 등을 공개 ", 1이면 "시드 자체를 공개" 등의 응답을 섞어서 보냄.
- 최종적으로는 서명 데이터 안에 여러 라운드의 (commit, response) 조합을 담게 된다

Algorithm 2: Sign(sk, Msg)

Input: sk: secret key $\mathtt{Seed}_{\mathtt{sk}} \in \{0,1\}^{2\lambda}$;
Msg $\in \{0,1\}^*$: message;
Output: Sgn: signature;
Data: λ: security parameter;
c: constant defined as $c = 2t - 1$;
$g \in \mathbb{F}_p^*$: generator of $\mathbb{E}$;
t: number of rounds;
w: weight of the second challenge;

// Expanding secret key

$\bar{\mathbf{e}}, \bar{\mathbf{e}}_G, \mathbf{H}, \overline{\mathbf{M}} \leftarrow \mathsf{ExpandSK}(\mathtt{Seed}_{\mathtt{sk}})$ | $\bar{\mathbf{e}}, \mathbf{H} \leftarrow \mathsf{ExpandSK}(\mathtt{Seed}_{\mathtt{sk}})$

// Computing the commitments

$\mathtt{Seed} \xleftarrow{\$} \{0,1\}^{\lambda}, \quad \mathtt{Salt} \xleftarrow{\$} \{0,1\}^{2\lambda}$
$(\mathtt{Seed}[1], \ldots, \mathtt{Seed}[t]) \leftarrow \mathsf{SeedLeaves}(\mathtt{Seed} \mid \mathtt{Salt})$
// Compute $\mathbf{v}[i]$ such that $\mathbf{v}[i] \star \mathbf{e}'[i] = \mathbf{e}$
for i **from** 1 **to** t **do**
$\bar{\mathbf{e}}'_G[i], \mathbf{u}'[i] \leftarrow \mathsf{CSPRNG}_{-\mathbb{F}_z^n \times \mathbb{F}_p^n}(\mathtt{Seed}[i] \mid \mathtt{Salt} \mid i+c)$ | $\bar{\mathbf{e}}'[i], \mathbf{u}'[i] \leftarrow \mathsf{CSPRNG}_{-\mathbb{F}_z^n \times \mathbb{F}_p^n}(\mathtt{Seed}[i] \mid \mathtt{Salt} \mid i+c)$
$\bar{\mathbf{v}}_G[i] \leftarrow \bar{\mathbf{e}}_G - \bar{\mathbf{e}}'_G[i]$
$\bar{\mathbf{e}}'[i] \leftarrow \bar{\mathbf{e}}'_G[i]\overline{\mathbf{M}}$
$\bar{\mathbf{v}}[i] \leftarrow \bar{\mathbf{e}} - \bar{\mathbf{e}}'[i]$
  for j **from** 1 **to** n **do**
    $\mathbf{v}[i]_j \leftarrow g^{\bar{\mathbf{v}}[i]_j}$
  $\mathbf{u}[i] \leftarrow \mathbf{v}[i] \star \mathbf{u}'[i]$
  $\mathbf{s}'[i] \leftarrow \mathbf{u}[i]\mathbf{H}^\top$
  $\mathtt{cmt}_0[i] \leftarrow \mathsf{Hash}(\mathbf{s}'[i] \mid \bar{\mathbf{v}}_G[i] \mid \mathtt{Salt} \mid i+c)$ | $\mathtt{cmt}_0[i] \leftarrow \mathsf{Hash}(\mathbf{s}'[i] \mid \bar{\mathbf{v}}[i] \mid \mathtt{Salt} \mid i+c)$
  $\mathtt{cmt}_1[i] \leftarrow \mathsf{Hash}(\mathtt{Seed}[i] \mid \mathtt{Salt} \mid i+c)$
$\mathtt{digest}_{\mathtt{cmt}_0} \leftarrow \mathsf{TreeRoot}(\mathtt{cmt}_0[1] \mid \cdots \mid \mathtt{cmt}_0[t])$
$\mathtt{digest}_{\mathtt{cmt}_1} \leftarrow \mathsf{Hash}(\mathtt{cmt}_1[1] \mid \cdots \mid \mathtt{cmt}_1[t])$
$\mathtt{digest}_{\mathtt{cmt}} \leftarrow \mathsf{Hash}(\mathtt{digest}_{\mathtt{cmt}_0} \mid \mathtt{digest}_{\mathtt{cmt}_1})$
// Computing first challenge
$\mathtt{digest}_{\mathtt{Msg}} \leftarrow \mathsf{Hash}(\mathtt{Msg})$
$\mathtt{digest}_{\mathtt{chall}_1} \leftarrow \mathsf{Hash}(\mathtt{digest}_{\mathtt{Msg}} \mid \mathtt{digest}_{\mathtt{cmt}} \mid \mathtt{Salt})$
$\mathtt{chall}_1 \leftarrow \mathsf{CSPRNG}_{-(\mathbb{F}_p^*)^t}(\mathtt{digest}_{\mathtt{chall}_1} \mid t+c)$
// Computing first response
for i **from** 1 **to** t **do**
  for j **from** 1 **to** n **do**
    $\mathbf{e}'[i]_j \leftarrow g^{\bar{\mathbf{e}}'[i]_j}$
  $\mathbf{y}[i] \leftarrow \mathbf{u}'[i] + \mathtt{chall}_1[i]\mathbf{e}'[i]$
// Computing second challenge
$\mathtt{digest}_{\mathtt{chall}_2} \leftarrow \mathsf{Hash}(\mathbf{y}[1] \mid \cdots \mid \mathbf{y}[t] \mid \mathtt{digest}_{\mathtt{chall}_1})$
$\mathtt{chall}_2 \leftarrow \mathsf{CSPRNG}_{-\mathcal{B}_{(t,w)}}(\mathtt{digest}_{\mathtt{chall}_2} \mid t+c+1)$
// Computing second response
$\mathtt{Proof} \leftarrow \mathsf{TreeProof}(\mathtt{cmt}_0[1] \mid \cdots \mid \mathtt{cmt}_0[t] \mid \mathtt{chall}_2)$
$\mathtt{Path} \leftarrow \mathsf{SeedPath}(\mathtt{Seed} \mid \mathtt{Salt} \mid \mathtt{chall}_2)$
for i **from** 1 **to** t **do**
  if $\mathtt{chall}_2[i] = 0$ **then**
    $\mathtt{resp}[i]_0 \leftarrow (\mathbf{y}[i], \bar{\mathbf{v}}_G[i])$ | $\mathtt{resp}[i]_0 \leftarrow (\mathbf{y}[i], \bar{\mathbf{v}}[i])$
    $\mathtt{resp}[i]_1 \leftarrow \mathtt{cmt}_1[i]$
// Assembling signature
$\mathtt{Sgn} \leftarrow (\mathtt{Salt}, \mathtt{digest}_{\mathtt{cmt}}, \mathtt{digest}_{\mathtt{chall}_2}, \mathtt{Path}, \mathtt{Proof}, \mathtt{resp})$
return Sgn

CROSS 검증

- 공개키 (Seedpk,s) 로부터 H를 재생성.
- 서명 속 Salt, resp, Path, Proof 등을 보고,
  서명자가 주장한 커밋과 실제 y, v가 일치하는지 등
  라운드별 검증을 수행.
- t개 라운드가 모두 합격이면 서명 유효.

Algorithm 3: Verify(pk, Msg, Sgn)

Input: pk: $(\mathtt{Seed}_{\mathtt{pk}}, \mathbf{s})$ public key;
$\mathtt{Msg} \in \{0,1\}^*$: message;
Sgn: $(\mathtt{Salt}, \mathtt{digest}_{\mathtt{cmt}}, \mathtt{digest}_{\mathtt{chall}_2}, \mathtt{Path}, \mathtt{Proof}, \mathtt{resp})$ signature;
Output: {True, False};
Data: λ: security parameter;
g: generator of $\mathbb{E}$;
t: number of rounds; w: weight of second challenge;
c: constant defined as $2t-1$;

// Recovering public key

1 $(\overline{\mathbf{W}}, \mathbf{V}) \leftarrow \mathsf{CSPRNG}_{-\mathbb{F}_z^{m\times(n-m)}\times\mathbb{F}_p^{(n-k)\times k}}(\mathtt{Seed}_{\mathtt{pk}} \mid 3t+2)$ | $\mathbf{V} \leftarrow \mathsf{CSPRNG}_{-\mathbb{F}_p^{(n-k)\times k}}(\mathtt{Seed}_{\mathtt{pk}} \mid 3t+2)$

2 $\mathbf{H} \leftarrow [\mathbf{V} \mid \mathrm{Id}_{n-k}]$
3 $\overline{\mathbf{M}} \leftarrow [\overline{\mathbf{W}} \mid \mathrm{Id}_m]$

// Computing challenges

4 $\mathtt{digest}_{\mathtt{Msg}} \leftarrow \mathsf{Hash}(\mathtt{Msg})$
5 $\mathtt{digest}_{\mathtt{chall}_1} \leftarrow \mathsf{Hash}(\mathtt{digest}_{\mathtt{Msg}} \mid \mathtt{digest}_{\mathtt{cmt}} \mid \mathtt{Salt})$
6 $\mathtt{chall}_1 \leftarrow \mathsf{CSPRNG}_{-(\mathbb{F}_p^*)^t}(\mathtt{digest}_{\mathtt{chall}_1} \mid t+c)$
7 $\mathtt{chall}_2 \leftarrow \mathsf{CSPRNG}_{-\mathcal{B}_{(t,w)}}(\mathtt{digest}_{\mathtt{chall}_2} \mid t+c+1)$

// Computing commitments

8 $(\mathsf{Seed}[i])_{i:\mathtt{chall}_2[i]=1} \leftarrow \mathsf{RebuildLeaves}(\mathtt{Path} \mid \mathtt{chall}_2 \mid \mathtt{Salt})$
9 **for** i **from** 1 **to** t **do**
10 **if** $\mathtt{chall}_2[i] = 1$ **then**
11 $\mathtt{cmt}_1[i] \leftarrow \mathsf{Hash}(\mathsf{Seed}[i] \mid \mathtt{Salt} \mid i+c)$
12 $\overline{\mathbf{e}}'_G[i], \mathbf{u}'[i] \leftarrow \mathsf{CSPRNG}_{-\mathbb{F}_z^m\times\mathbb{F}_p^n}(\mathsf{Seed}[i] \mid \mathtt{Salt} \mid i+c)$ | $\overline{\mathbf{e}}'[i], \mathbf{u}'[i] \leftarrow \mathsf{CSPRNG}_{-\mathbb{F}_z^n\times\mathbb{F}_p^n}(\mathsf{Seed}[i] \mid \mathtt{Salt} \mid i+c)$
$\overline{\mathbf{e}}'[i] \leftarrow \overline{\mathbf{e}}'_G[i]\overline{\mathbf{M}}$
13 **for** j **from** 1 **to** n **do**
$\mathbf{e}'[i]_j \leftarrow g^{\overline{\mathbf{e}}[i]_j}$
14 $\mathbf{y}[i] \leftarrow \mathbf{u}'[i] + \mathtt{chall}_1[i]\mathbf{e}'[i]$
15 **if** $\mathtt{chall}_2[i] = 0$ **then**
16 $\mathtt{cmt}_1[i] \leftarrow \mathtt{resp}[i]_1$
17 $(\mathbf{y}[i], \overline{\mathbf{v}}_G[i]) \leftarrow \mathtt{resp}[i]_0$ | $(\mathbf{y}[i], \overline{\mathbf{v}}[i]) \leftarrow \mathtt{resp}[i]_0$
Check if $\overline{\mathbf{v}}_G[i] \in \mathbb{F}_z^m$ | Check if $\overline{\mathbf{v}}[i] \in \mathbb{F}_z^n$
$\overline{\mathbf{v}}[i] \leftarrow \overline{\mathbf{v}}_G[i]\overline{\mathbf{M}}$
18 **for** j **from** 1 **to** n **do**
$\mathbf{v}[i]_j \leftarrow g^{\overline{\mathbf{v}}[i]_j}$
19 $\mathbf{y}'[i] \leftarrow \mathbf{v}[\mathbf{i}] \star \mathbf{y}[i]$
20 $\mathbf{s}'[i] \leftarrow \mathbf{y}'[i]\mathbf{H}^\top - \mathtt{chall}_1[i]\mathbf{s}$
21 $\mathtt{cmt}_0[i] \leftarrow \mathsf{Hash}(\mathbf{s}'[i] \mid \overline{\mathbf{v}}_G[i] \mid \mathtt{Salt} \mid i+c)$ | $\mathtt{cmt}_0[i] \leftarrow \mathsf{Hash}(\mathbf{s}'[i] \mid \overline{\mathbf{v}}[i] \mid \mathtt{Salt} \mid i+c)$

// Checking digests

22 $\mathtt{digest}_{\mathtt{cmt}_0} \leftarrow \mathsf{RecomputeRoot}(\mathtt{cmt}_0 \mid \mathsf{Proof} \mid \mathtt{chall}_2)$
23 $\mathtt{digest}_{\mathtt{cmt}_1} \leftarrow \mathsf{Hash}(\mathtt{cmt}_1[1] \mid \cdots \mid \mathtt{cmt}_1[t])$
24 $\mathtt{digest}'_{\mathtt{cmt}} \leftarrow \mathsf{Hash}(\mathtt{digest}_{\mathtt{cmt}_0} \mid \mathtt{digest}_{\mathtt{cmt}_1})$
25 $\mathtt{digest}'_{\mathtt{chall}_2} \leftarrow \mathsf{Hash}(\mathbf{y}[1] \mid \cdots \mid \mathbf{y}[t] \mid \mathtt{digest}_{\mathtt{chall}_1})$
26 **if** $\mathtt{digest}_{\mathtt{cmt}} = \mathtt{digest}'_{\mathtt{cmt}}$ **and** $\mathtt{digest}_{\mathtt{chall}_2} = \mathtt{digest}'_{\mathtt{chall}_2}$ **then**
27 **return** True
28 **return** False

향후 계획

- **코드 기반 암호 최적화…예정**
- CROSS 코드 분석
- CROSS 최적화
  - AES+Keccak pqclean에 공개된 최적 구현 포함
- LESS 코드 분석
- LESS 에서 공개한 NEON 코드 분석
  - 어셈블리 구현이 아닌 ARM에서 제공하는 NEON intrinsics 만을 사용한 구현
- LESS 최적화
  - AES+Keccak pqclean에 공개된 최적 구현 포함

```
/*
 * Fix width 16-bit Barrett multiplication Q = 127
 * c = (a * b) % q
 */
#define barrett_mul_u16(c, a, b, t)        \
    vmul_lo16(a, a, b); /* lo = (a * b)  */  \
    vsr16(t, a, 7);     /* hi = (lo >> 7) */ \
    vadd16(a, a, t);    /* lo = (lo + hi) */ \
    vsl16(t, t, 7);     /* hi = (hi << 7) */ \
    vsub16(c, a, t);    /* c  = (lo - hi) */
```

https://github.com/less-sig/LESS/blob/main/Optimized_Implementation/neon/lib/transpose_neon.c

Q & A