LESS: Digital Signatures from Linear Code Equivalence

2nd Oxford Post-Quantum Cryptography Summit

Marco Baldi, Alessandro Barenghi, Luke Beckwith, Jean-François Biasse, Andre Esser, Kris Gaj, Kamyar Mohajerani, Gerardo Pelosi, **Edoardo Persichetti**, Markku-J. O. Saarinen, Paolo Santini, Robert Wallace

5 September 2023

DEPARTMENT OF MATHEMATICAL SCIENCES
Charles E. Schmidt College of Science
Florida Atlantic University

In This Talk

Roadmap

- Motivation and Background
- Code Equivalence
- LESS
- Considerations

Roadmap

- Motivation and Background
- Code Equivalence
- LESS
- Considerations

$[n, k]$ Linear Code over $\mathbb{F}_q$

A subspace of dimension k of $\mathbb{F}_q^n$. Value n is called length.

Hamming Metric

$wt(x) = |\{i : x_i \neq 0, 1 \leq i \leq n\}|$, $d(x, y) = wt(x - y)$.
Minimum distance (of $\mathfrak{C}$): $\min\{d(x, y) : x, y \in \mathfrak{C}\}$.

Generator Matrix

$G \in \mathbb{F}_q^{k \times n}$ defines the code as : $x \in \mathfrak{C} \iff x = uG$ for $u \in \mathbb{F}_q^k$.
Not unique: $SG, S \in GL(k, q)$; Systematic form: $(I_k | M)$.

Parity-check Matrix

$H \in \mathbb{F}_q^{(n-k) \times n}$ defines the code as: $x \in \mathfrak{C} \iff Hx^T = 0$ (syndrome).
Not unique: $SH, S \in GL(n - k, q)$; Systematic form: $(M^T | I_{n-k})$.

w-error correcting: $\exists$ algorithm that corrects up to w errors.

Use hard problems from coding theory, such as SDP in the Hamming metric.

Traditional Code-Based Cryptography

Use hard problems from coding theory, such as SDP in the Hamming metric.

For encryption, one can obtain a trapdoor by masking the private code.

Traditional Code-Based Cryptography

Use hard problems from coding theory, such as SDP in the Hamming metric.

For encryption, one can obtain a trapdoor by masking the private code.

Example (McEliece/Niederreiter): use change of basis S and permutation P to obtain equivalent code.

Use hard problems from coding theory, such as SDP in the Hamming metric.

For encryption, one can obtain a trapdoor by masking the private code.

Example (McEliece/Niederreiter): use change of basis S and permutation P to obtain equivalent code.

Hardness is an assumption which depends on chosen code family.

Use hard problems from coding theory, such as SDP in the Hamming metric.

For encryption, one can obtain a trapdoor by masking the private code.

Example (McEliece/Niederreiter): use change of basis S and permutation P to obtain equivalent code.

Hardness is an assumption which depends on chosen code family.

This works well for encryption...

(Classic McEliece, BIKE, HQC)

Use hard problems from coding theory, such as SDP in the Hamming metric.

For encryption, one can obtain a trapdoor by masking the private code.

Example (McEliece/Niederreiter): use change of basis S and permutation P to obtain equivalent code.

Hardness is an assumption which depends on chosen code family.

This works well for encryption...

(Classic McEliece, BIKE, HQC)

...far less so for signature schemes.

(CFS, KKS, Stern,...)

Use hard problems from coding theory, such as SDP in the Hamming metric.

For encryption, one can obtain a trapdoor by masking the private code.

Example (McEliece/Niederreiter): use change of basis S and permutation P to obtain equivalent code.

Hardness is an assumption which depends on chosen code family.

This works well for encryption...
(Classic McEliece, BIKE, HQC)

...far less so for signature schemes.
(CFS, KKS, Stern,...)

History suggest that we have to do things a little differently.

Roadmap

- Motivation and Background
- Code Equivalence
- LESS
- Considerations

McEliece security requires to keep the private code secret (!); but, what is the hardness of finding the mask itself?

McEliece security requires to keep the private code secret (!); but, what is the hardness of finding the mask itself?

The pair (S, P) is an isometry for the Hamming metric, as it preserves the weight distribution of the code.

McEliece security requires to keep the private code secret (!); but, what is the hardness of finding the mask itself?

The pair (S, P) is an isometry for the Hamming metric, as it preserves the weight distribution of the code.

The problem of finding such a map is well-known in coding theory; indeed, it is a kind of isomorphism problem which recalls other similar ones (isomorphism of polynomials, isogenies, etc.)

Going Solo

McEliece security requires to keep the private code secret (!); but, what is the hardness of finding the mask itself?

The pair (S, P) is an isometry for the Hamming metric, as it preserves the weight distribution of the code.

The problem of finding such a map is well-known in coding theory; indeed, it is a kind of isomorphism problem which recalls other similar ones (isomorphism of polynomials, isogenies, etc.)

Could Code Equivalence be used as a stand-alone problem?

McEliece security requires to keep the private code secret (!); but, what is the hardness of finding the mask itself?

The pair (S, P) is an isometry for the Hamming metric, as it preserves the weight distribution of the code.

The problem of finding such a map is well-known in coding theory; indeed, it is a kind of isomorphism problem which recalls other similar ones (isomorphism of polynomials, isogenies, etc.)

Could Code Equivalence be used as a stand-alone problem?

Possible to construct a ZK protocol based exclusively on the hardness of the code equivalence problem (then, apply Fiat-Shamir).

(Biasse, Micheli, P., Santini, 2020)

Isometries in the Hamming Metric

2 Code Equivalence

Three types:

- Permutations: $\pi\big((a_1, a_2, \ldots, a_n)\big) = \big(a_{\pi(1)}, a_{\pi(2)}, \ldots, a_{\pi(n)}\big).$

Three types:

- Permutations: $\pi\big((a_1, a_2, \ldots, a_n)\big) = \big(a_{\pi(1)}, a_{\pi(2)}, \ldots, a_{\pi(n)}\big)$.
- Monomials: permutations + scaling factors: $\mu = (v; \pi)$, with $v \in (\mathbb{F}_q^*)^n$

$$\mu\big((a_1, a_2, \ldots, a_n)\big) = \big(v_1 \cdot a_{\pi(1)}, v_2 \cdot a_{\pi(2)}, \ldots, v_n \cdot a_{\pi(n)}\big)$$

Monomial matrix: permutation $\times$ diagonal.

Isometries in the Hamming Metric

Three types:

- Permutations: $\pi\big((a_1, a_2, \ldots, a_n)\big) = \big(a_{\pi(1)}, a_{\pi(2)}, \ldots, a_{\pi(n)}\big)$.
- Monomials: permutations + scaling factors: $\mu = (v; \pi)$, with $v \in (\mathbb{F}_q^*)^n$

  $$\mu\big((a_1, a_2, \ldots, a_n)\big) = \big(v_1 \cdot a_{\pi(1)}, v_2 \cdot a_{\pi(2)}, \ldots, v_n \cdot a_{\pi(n)}\big)$$

  Monomial matrix: permutation $\times$ diagonal.
- Monomials + field automorphism (we usually ignore this in cryptography).

Three types:

- Permutations: $\pi\big((a_1, a_2, \dots, a_n)\big) = \big(a_{\pi(1)}, a_{\pi(2)}, \dots, a_{\pi(n)}\big)$.
- Monomials: permutations + scaling factors: $\mu = (v; \pi)$, with $v \in (\mathbb{F}_q^*)^n$

$$\mu\big((a_1, a_2, \dots, a_n)\big) = \big(v_1 \cdot a_{\pi(1)}, v_2 \cdot a_{\pi(2)}, \dots, v_n \cdot a_{\pi(n)}\big)$$

  Monomial matrix: permutation $\times$ diagonal.
- Monomials + field automorphism (we usually ignore this in cryptography).

Can be seen as a group action on linear codes by setting:

Three types:

- Permutations: $\pi\big((a_1, a_2, \dots, a_n)\big) = \big(a_{\pi(1)}, a_{\pi(2)}, \dots, a_{\pi(n)}\big)$.
- Monomials: permutations + scaling factors: $\mu = (v; \pi)$, with $v \in (\mathbb{F}_q^*)^n$

  $$\mu\big((a_1, a_2, \dots, a_n)\big) = \big(v_1 \cdot a_{\pi(1)}, v_2 \cdot a_{\pi(2)}, \dots, v_n \cdot a_{\pi(n)}\big)$$

  Monomial matrix: permutation $\times$ diagonal.
- Monomials + field automorphism (we usually ignore this in cryptography).

Can be seen as a group action on linear codes by setting:

- $\mathcal{G} = M_n(q)$, the monomial group;

Three types:

- Permutations: $\pi\big((a_1, a_2, \ldots, a_n)\big) = \big(a_{\pi(1)}, a_{\pi(2)}, \ldots, a_{\pi(n)}\big)$.
- Monomials: permutations + scaling factors: $\mu = (v; \pi)$, with $v \in (\mathbb{F}_q^*)^n$

  $$\mu\big((a_1, a_2, \ldots, a_n)\big) = \big(v_1 \cdot a_{\pi(1)}, v_2 \cdot a_{\pi(2)}, \ldots, v_n \cdot a_{\pi(n)}\big)$$

  Monomial matrix: permutation $\times$ diagonal.
- Monomials + field automorphism (we usually ignore this in cryptography).

Can be seen as a group action on linear codes by setting:

- $\mathcal{G} = M_n(q)$, the monomial group;
- $\mathcal{X} \subseteq \mathbb{F}_q^{k\times n}$, the set of generator matrices in RREF.

Three types:

- Permutations: $\pi\big((a_1, a_2, \ldots, a_n)\big) = \big(a_{\pi(1)}, a_{\pi(2)}, \ldots, a_{\pi(n)}\big)$.
- Monomials: permutations + scaling factors: $\mu = (v; \pi)$, with $v \in (\mathbb{F}_q^*)^n$

$$\mu\big((a_1, a_2, \ldots, a_n)\big) = \big(v_1 \cdot a_{\pi(1)}, v_2 \cdot a_{\pi(2)}, \ldots, v_n \cdot a_{\pi(n)}\big)$$

  Monomial matrix: permutation $\times$ diagonal.
- Monomials + field automorphism (we usually ignore this in cryptography).

Can be seen as a group action on linear codes by setting:

- $\mathcal{G} = M_n(q)$, the monomial group;
- $\mathcal{X} \subseteq \mathbb{F}_q^{k\times n}$, the set of generator matrices in RREF.

Code-based Group Action

$$\begin{array}{cccc} \star: & \mathcal{X} \times \mathcal{G} & \to & \mathcal{X} \\ & (G_0, Q) & \mapsto & RREF(G_0 \cdot Q) \end{array}$$

Two codes are equivalent if they are connected by an isometry.

Two codes are equivalent if they are connected by an isometry.

We talk about permutation, linear (and semilinear) equivalence, respectively.

Two codes are equivalent if they are connected by an isometry.

We talk about permutation, linear (and semilinear) equivalence, respectively.

Deciding whether two codes are equivalent is known as the code equivalence problem, according to the chosen notion of isometry.

Two codes are equivalent if they are connected by an isometry.

We talk about permutation, linear (and semilinear) equivalence, respectively.

Deciding whether two codes are equivalent is known as the code equivalence problem, according to the chosen notion of isometry.

The vectorization problem for our group action is the computational version of code equivalence.

Two codes are equivalent if they are connected by an isometry.

We talk about permutation, linear (and semilinear) equivalence, respectively.

Deciding whether two codes are equivalent is known as the code equivalence problem, according to the chosen notion of isometry.

The vectorization problem for our group action is the computational version of code equivalence.

Linear Equivalence Problem (LEP)

Given $\mathfrak{C}_0, \mathfrak{C}_1 \subseteq \mathbb{F}_q^n$, find a monomial μ such that $\mu(\mathfrak{C}_0) = \mathfrak{C}_1$.
Equivalently, given generators $G_0, G_1 \in \mathbb{F}_q^{k\times n}$, find $Q \in M_n(q)$ such that

$$G_1 = RREF(G_0 Q).$$

Roadmap

- Motivation and Background
- Code Equivalence
- LESS
- Considerations

Select hash function $\mathbf{H}$.

Select hash function $\mathbf{H}$.

Key Generation

- Choose random q-ary code $\mathfrak{C}$, given by generator matrix G_0.
- sk: monomial matrix Q.
- pk: matrix $G_1 = RREF(G_0Q)$.

The LESS ZKID

Select hash function $\mathbf{H}$.

Key Generation

- Choose random q-ary code $\mathfrak{C}$, given by generator matrix G_0.
- sk: monomial matrix Q.
- pk: matrix $G_1 = RREF(G_0 Q)$.

Prover		**Verifier**
Choose random monomial matrix $\tilde{Q} \in M_n(q)$.		
Compute $\tilde{G} = RREF(G_0 \tilde{Q})$.		
Set $cmt = \mathbf{H}(\tilde{G})$	$\xrightarrow{cmt}$	
	$\xleftarrow{b}$	Select random $b \in \{0, 1\}$.
If $b = 0$ set $rsp = \tilde{Q}$	$\xrightarrow{rsp}$	Verify $\mathbf{H}(RREF(G_0 \cdot rsp)) = cmt$.
If $b = 1$ set $rsp = Q^{-1}\tilde{Q}$		Verify $\mathbf{H}(RREF(G_1 \cdot rsp)) = cmt$.

It is easy to prove completeness, 2-special soundness and honest-verifier zero-knowledge.

It is easy to prove completeness, 2-special soundness and honest-verifier zero-knowledge.

Before Fiat-Shamir, reduce soundness error $1/2 \implies t = \lambda$ parallel repetitions.

It is easy to prove completeness, 2-special soundness and honest-verifier zero-knowledge.

Before Fiat-Shamir, reduce soundness error $1/2 \implies t = \lambda$ parallel repetitions.

The protocol can be greatly improved with the following modifications:

(Barenghi, Biasse, P., Santini, 2021)

It is easy to prove completeness, 2-special soundness and honest-verifier zero-knowledge.

Before Fiat-Shamir, reduce soundness error $1/2 \implies t = \lambda$ parallel repetitions.

The protocol can be greatly improved with the following modifications:

(Barenghi, Biasse, P., Santini, 2021)

- Use multiple public keys and non-binary challenges.

$+$ Lower soundness error: $1/2 \to 1/2^{\ell}$.

$-$ Rapid increase in public key size.

It is easy to prove completeness, 2-special soundness and honest-verifier zero-knowledge.

Before Fiat-Shamir, reduce soundness error $1/2 \implies t = \lambda$ parallel repetitions.

The protocol can be greatly improved with the following modifications:

(Barenghi, Biasse, P., Santini, 2021)

- Use multiple public keys and non-binary challenges.

$+$ Lower soundness error: $1/2 \to 1/2^\ell$.

$-$ Rapid increase in public key size.

- Use a fixed-weight challenge string.

$+$ Exploits imbalance in cost of response: seed vs monomial.

$-$ Larger number of iterations.

LESS Signatures

It is easy to prove completeness, 2-special soundness and honest-verifier zero-knowledge.

Before Fiat-Shamir, reduce soundness error $1/2 \implies t = \lambda$ parallel repetitions.

The protocol can be greatly improved with the following modifications:
(Barenghi, Biasse, P., Santini, 2021)

- Use multiple public keys and non-binary challenges.

+ Lower soundness error: $1/2 \to 1/2^{\ell}$.

− Rapid increase in public key size.

- Use a fixed-weight challenge string.

+ Exploits imbalance in cost of response: seed vs monomial.

− Larger number of iterations.

Such modifications do not affect security, only requiring small tweaks in proofs or switching to equivalent security assumptions.

Roadmap

- Motivation and Background
- Code Equivalence
- LESS
- Considerations

PEP is not NP-complete, unless the polynomial hierarchy collapses.

(Petrank, Roth, 1997)

PEP is not NP-complete, unless the polynomial hierarchy collapses.

(Petrank, Roth, 1997)

PEP is also deeply connected with Graph Isomorphism (GI) (reductions in both ways!), solvable in quasi-polynomial time.

PEP is not NP-complete, unless the polynomial hierarchy collapses.

(Petrank, Roth, 1997)

PEP is also deeply connected with Graph Isomorphism (GI) (reductions in both ways!), solvable in quasi-polynomial time.

At the same time, PEP is “not necessarily easy”.

(Petrank, Roth, 1997)

PEP is not NP-complete, unless the polynomial hierarchy collapses.

(Petrank, Roth, 1997)

PEP is also deeply connected with Graph Isomorphism (GI) (reductions in both ways!), solvable in quasi-polynomial time.

At the same time, PEP is “not necessarily easy”.

(Petrank, Roth, 1997)

PEP is a special case of LEP; indeed, with time $O(q)$, we have

$$PEP \xleftarrow{\text{Reduces to}} LEP$$

PEP is not NP-complete, unless the polynomial hierarchy collapses.
(Petrank, Roth, 1997)

PEP is also deeply connected with Graph Isomorphism (GI) (reductions in both ways!), solvable in quasi-polynomial time.

At the same time, PEP is “not necessarily easy”.
(Petrank, Roth, 1997)

PEP is a special case of LEP; indeed, with time $O(q)$, we have

$$PEP \xleftarrow{\text{Reduces to}} LEP$$

As a consequence, most solvers for PEP can be easily adapted to solve LEP as well.

Attack Strategy 1: Weak Instances

Exploit a variety of properties, give rise to (potentially) most efficient solvers.

Attack Strategy 1: Weak Instances

Exploit a variety of properties, give rise to (potentially) most efficient solvers.

- Support Splitting Algorithm (SSA) looks for invariants to distinguish equivalent codes.
  (Sendrier, 2000)

Attack Strategy 1: Weak Instances

Exploit a variety of properties, give rise to (potentially) most efficient solvers.

- Support Splitting Algorithm (SSA) looks for invariants to distinguish equivalent codes. (Sendrier, 2000)

  Weight Enumerator Function (WEF) is one, but too expensive; compute on hull.

$$\mathcal{H}(\mathfrak{C}) = \mathfrak{C} \cap \mathfrak{C}^{\perp}$$

Attack Strategy 1: Weak Instances

Exploit a variety of properties, give rise to (potentially) most efficient solvers.

- Support Splitting Algorithm (SSA) looks for invariants to distinguish equivalent codes.

  (Sendrier, 2000)

  Weight Enumerator Function (WEF) is one, but too expensive; compute on hull.

$$\mathcal{H}(\mathfrak{C}) = \mathfrak{C} \cap \mathfrak{C}^{\perp}$$

  If $\mathfrak{C}_1 = \pi(\mathfrak{C}_0)$, then $\mathcal{H}(\mathfrak{C}_1) = \pi\big(\mathcal{H}(\mathfrak{C}_0)\big)$; running in $\mathcal{O}(q^h)$.

Attack Strategy 1: Weak Instances

Exploit a variety of properties, give rise to (potentially) most efficient solvers.

- Support Splitting Algorithm (SSA) looks for invariants to distinguish equivalent codes. (Sendrier, 2000)

  Weight Enumerator Function (WEF) is one, but too expensive; compute on hull.

  $$\mathcal{H}(\mathfrak{C}) = \mathfrak{C} \cap \mathfrak{C}^{\perp}$$

  If $\mathfrak{C}_1 = \pi(\mathfrak{C}_0)$, then $\mathcal{H}(\mathfrak{C}_1) = \pi\big(\mathcal{H}(\mathfrak{C}_0)\big)$; running in $\mathcal{O}(q^h)$.

  Random codes tend to have small hulls, which makes attack practical.

Attack Strategy 1: Weak Instances

Exploit a variety of properties, give rise to (potentially) most efficient solvers.

- Support Splitting Algorithm (SSA) looks for invariants to distinguish equivalent codes.
  (Sendrier, 2000)

  Weight Enumerator Function (WEF) is one, but too expensive; compute on hull.

  $$\mathcal{H}(\mathfrak{C}) = \mathfrak{C} \cap \mathfrak{C}^{\perp}$$

  If $\mathfrak{C}_1 = \pi(\mathfrak{C}_0)$, then $\mathcal{H}(\mathfrak{C}_1) = \pi\big(\mathcal{H}(\mathfrak{C}_0)\big)$; running in $\mathcal{O}(q^h)$.

  Random codes tend to have small hulls, which makes attack practical.
  - Use (weakly) self-dual codes to avoid attack.

Attack Strategy 1: Weak Instances

Exploit a variety of properties, give rise to (potentially) most efficient solvers.

- Support Splitting Algorithm (SSA) looks for invariants to distinguish equivalent codes. (Sendrier, 2000)

  Weight Enumerator Function (WEF) is one, but too expensive; compute on hull.

  $$\mathcal{H}(\mathfrak{C}) = \mathfrak{C} \cap \mathfrak{C}^{\perp}$$

  If $\mathfrak{C}_1 = \pi(\mathfrak{C}_0)$, then $\mathcal{H}(\mathfrak{C}_1) = \pi\big(\mathcal{H}(\mathfrak{C}_0)\big)$; running in $\mathcal{O}(q^h)$.

  Random codes tend to have small hulls, which makes attack practical.

  * Use (weakly) self-dual codes to avoid attack.
  * To solve LEP, need to target closure of the code; these are always self-dual for $q \geq 5$.

Attack Strategy 1: Weak Instances

Exploit a variety of properties, give rise to (potentially) most efficient solvers.

- Support Splitting Algorithm (SSA) looks for invariants to distinguish equivalent codes. (Sendrier, 2000)

  Weight Enumerator Function (WEF) is one, but too expensive; compute on hull.

  $$\mathcal{H}(\mathfrak{C}) = \mathfrak{C} \cap \mathfrak{C}^{\perp}$$

  If $\mathfrak{C}_1 = \pi(\mathfrak{C}_0)$, then $\mathcal{H}(\mathfrak{C}_1) = \pi\big(\mathcal{H}(\mathfrak{C}_0)\big)$; running in $\mathcal{O}(q^h)$.

  Random codes tend to have small hulls, which makes attack practical.
  * Use (weakly) self-dual codes to avoid attack.
  * To solve LEP, need to target closure of the code; these are always self-dual for $q \geq 5$.

- Algebraic approaches of different nature, for example:

Attack Strategy 1: Weak Instances

Exploit a variety of properties, give rise to (potentially) most efficient solvers.

- Support Splitting Algorithm (SSA) looks for invariants to distinguish equivalent codes. (Sendrier, 2000)

  Weight Enumerator Function (WEF) is one, but too expensive; compute on hull.

  $$\mathcal{H}(\mathfrak{C}) = \mathfrak{C} \cap \mathfrak{C}^{\perp}$$

  If $\mathfrak{C}_1 = \pi(\mathfrak{C}_0)$, then $\mathcal{H}(\mathfrak{C}_1) = \pi\big(\mathcal{H}(\mathfrak{C}_0)\big)$; running in $\mathcal{O}(q^h)$.

  Random codes tend to have small hulls, which makes attack practical.
    * Use (weakly) self-dual codes to avoid attack.
    * To solve LEP, need to target closure of the code; these are always self-dual for $q \geq 5$.

- Algebraic approaches of different nature, for example:
    * Set up a system of equations, solve via Gröbner basis. (Saeed-Taha, 2017)

Attack Strategy 1: Weak Instances

Exploit a variety of properties, give rise to (potentially) most efficient solvers.

- Support Splitting Algorithm (SSA) looks for invariants to distinguish equivalent codes. (Sendrier, 2000)

  Weight Enumerator Function (WEF) is one, but too expensive; compute on hull.

  $$\mathcal{H}(\mathfrak{C}) = \mathfrak{C} \cap \mathfrak{C}^{\perp}$$

  If $\mathfrak{C}_1 = \pi(\mathfrak{C}_0)$, then $\mathcal{H}(\mathfrak{C}_1) = \pi\big(\mathcal{H}(\mathfrak{C}_0)\big)$; running in $\mathcal{O}(q^h)$.

  Random codes tend to have small hulls, which makes attack practical.

  * Use (weakly) self-dual codes to avoid attack.
  * To solve LEP, need to target closure of the code; these are always self-dual for $q \geq 5$.

- Algebraic approaches of different nature, for example:
  * Set up a system of equations, solve via Gröbner basis. (Saeed-Taha, 2017)
  * Exploit reduction to graph isomorphism. (Bardet et al., 2020)

Attack Strategy 1: Weak Instances

Exploit a variety of properties, give rise to (potentially) most efficient solvers.

- Support Splitting Algorithm (SSA) looks for invariants to distinguish equivalent codes. (Sendrier, 2000)

  Weight Enumerator Function (WEF) is one, but too expensive; compute on hull.

  $$\mathcal{H}(\mathfrak{C}) = \mathfrak{C} \cap \mathfrak{C}^{\perp}$$

  If $\mathfrak{C}_1 = \pi(\mathfrak{C}_0)$, then $\mathcal{H}(\mathfrak{C}_1) = \pi\big(\mathcal{H}(\mathfrak{C}_0)\big)$; running in $\mathcal{O}(q^h)$.

  Random codes tend to have small hulls, which makes attack practical.
  - Use (weakly) self-dual codes to avoid attack.
  - To solve LEP, need to target closure of the code; these are always self-dual for $q \geq 5$.

- Algebraic approaches of different nature, for example:
  - Set up a system of equations, solve via Gröbner basis. (Saeed-Taha, 2017)
  - Exploit reduction to graph isomorphism. (Bardet et al., 2020)

  These are only efficient (or applicable in the first place) if hull is trivial.

Attack Strategy 2: Codeword Search

Action of π can be guessed from the set of all codewords with small weight w. (Leon, 1982)

Action of π can be guessed from the set of all codewords with small weight w. (Leon, 1982)

Moderate w guarantees no spurious solution and sufficiently low number of codewords.

Attack Strategy 2: Codeword Search

Action of π can be guessed from the set of all codewords with small weight w. (Leon, 1982)

Moderate w guarantees no spurious solution and sufficiently low number of codewords.

In practice, minimum distance plus 1 or 2 is enough to guarantee enough structure.

Attack Strategy 2: Codeword Search

Action of π can be guessed from the set of all codewords with small weight w. (Leon, 1982)

Moderate w guarantees no spurious solution and sufficiently low number of codewords.

In practice, minimum distance plus 1 or 2 is enough to guarantee enough structure.

The attack then consists of:

Attack Strategy 2: Codeword Search

Action of π can be guessed from the set of all codewords with small weight w. (Leon, 1982)

Moderate w guarantees no spurious solution and sufficiently low number of codewords.

In practice, minimum distance plus 1 or 2 is enough to guarantee enough structure.

The attack then consists of:

- Finding codewords (use ISD).

Attack Strategy 2: Codeword Search

Action of π can be guessed from the set of all codewords with small weight w. (Leon, 1982)

Moderate w guarantees no spurious solution and sufficiently low number of codewords.

In practice, minimum distance plus 1 or 2 is enough to guarantee enough structure.

The attack then consists of:

- Finding codewords (use ISD).
- Matching to extract permutation.

Attack Strategy 2: Codeword Search

Action of π can be guessed from the set of all codewords with small weight w. (Leon, 1982)

Moderate w guarantees no spurious solution and sufficiently low number of codewords.

In practice, minimum distance plus 1 or 2 is enough to guarantee enough structure.

The attack then consists of:

- Finding codewords (use ISD).
- Matching to extract permutation.

Cost is $\approx 2\log(N_w)C_{\text{isd}}(n, k, q, w)$ + linear algebra.

Attack Strategy 2: Codeword Search

Action of π can be guessed from the set of all codewords with small weight w. (Leon, 1982)

Moderate w guarantees no spurious solution and sufficiently low number of codewords.

In practice, minimum distance plus 1 or 2 is enough to guarantee enough structure.

The attack then consists of:

- Finding codewords (use ISD).
- Matching to extract permutation.

Cost is $\approx 2\log(N_w) C_{\text{isd}}(n, k, q, w)$ + linear algebra.

Permutations preserve multiset of entries $\implies$ no need to find all words of weight w.
(Beullens, 2020)

Attack Strategy 2: Codeword Search

Action of π can be guessed from the set of all codewords with small weight w. (Leon, 1982)

Moderate w guarantees no spurious solution and sufficiently low number of codewords.

In practice, minimum distance plus 1 or 2 is enough to guarantee enough structure.

The attack then consists of:

- Finding codewords (use ISD).
- Matching to extract permutation.

Cost is $\approx 2\log(N_w)C_{\text{isd}}(n, k, q, w)$ + linear algebra.

Permutations preserve multiset of entries $\implies$ no need to find all words of weight w. (Beullens, 2020)

Probabilistic algorithm, advantageous when q is large.

Design Considerations

We parametrize using latter type of attacks, following conservative criterion. Namely, we pick n, k, q so that, for any d and any w, we have:

$$\frac{1}{\sqrt{N_d(w)}} \cdot C_{\mathrm{ISD}}^{(d)}(n, k, q, w) > 2^{\lambda}.$$

We parametrize using latter type of attacks, following conservative criterion. Namely, we pick n, k, q so that, for any d and any w, we have:

$$\frac{1}{\sqrt{N_d(w)}} \cdot C_{\mathsf{ISD}}^{(d)}(n, k, q, w) > 2^{\lambda}.$$

The design of LESS allows for high degree of flexibility and customizable features according to goal.

We parametrize using latter type of attacks, following conservative criterion. Namely, we pick n, k, q so that, for any d and any w, we have:

$$\frac{1}{\sqrt{N_d(w)}} \cdot C_{\text{ISD}}^{(d)}(n, k, q, w) > 2^{\lambda}.$$

The design of LESS allows for high degree of flexibility and customizable features according to goal.

We select two parameter sets per category level (plus a third at level 1):

Design Considerations

We parametrize using latter type of attacks, following conservative criterion. Namely, we pick n, k, q so that, for any d and any w, we have:

$$\frac{1}{\sqrt{N_d(w)}} \cdot C_{\mathsf{ISD}}^{(d)}(n, k, q, w) > 2^{\lambda}.$$

The design of LESS allows for high degree of flexibility and customizable features according to goal.

We select two parameter sets per category level (plus a third at level 1):

- Balanced: yields similar sizes for PK and signature, e.g. minimizing their sum.

Design Considerations

We parametrize using latter type of attacks, following conservative criterion. Namely, we pick n, k, q so that, for any d and any w, we have:

$$\frac{1}{\sqrt{N_d(w)}} \cdot C_{\mathsf{ISD}}^{(d)}(n, k, q, w) > 2^{\lambda}.$$

The design of LESS allows for high degree of flexibility and customizable features according to goal.

We select two parameter sets per category level (plus a third at level 1):

- Balanced: yields similar sizes for PK and signature, e.g. minimizing their sum.
- Short: sacrifices PK size to push for smallest signature.

Design Considerations

We parametrize using latter type of attacks, following conservative criterion. Namely, we pick n, k, q so that, for any d and any w, we have:

$$\frac{1}{\sqrt{N_d(w)}} \cdot C_{\mathsf{ISD}}^{(d)}(n,k,q,w) > 2^{\lambda}.$$

The design of LESS allows for high degree of flexibility and customizable features according to goal.

We select two parameter sets per category level (plus a third at level 1):

- Balanced: yields similar sizes for PK and signature, e.g. minimizing their sum.
- Short: sacrifices PK size to push for smallest signature.

We use SHAKE as our PRNG and SHA-3 for the collision-resistant hash function.

We parametrize using latter type of attacks, following conservative criterion. Namely, we pick n, k, q so that, for any d and any w, we have:

$$\frac{1}{\sqrt{N_d(w)}} \cdot C_{\mathsf{ISD}}^{(d)}(n, k, q, w) > 2^{\lambda}.$$

The design of LESS allows for high degree of flexibility and customizable features according to goal.

We select two parameter sets per category level (plus a third at level 1):

- Balanced: yields similar sizes for PK and signature, e.g. minimizing their sum.
- Short: sacrifices PK size to push for smallest signature.

We use SHAKE as our PRNG and SHA-3 for the collision-resistant hash function.

We compactly generate and transmit seeds using a seed tree structure.

Performance

Protocol parameters (t, w, s) infer performance profile:

Performance

Protocol parameters (t, w, s) infer performance profile:

$$pk = (s-1)\ell_{G_i} + \ell_{seed}$$
$$sig = \ell_{salt} + \ell_{seed\_tree} + w \cdot \ell_{mono} + \ell_{digest}$$

Performance

Protocol parameters (t, w, s) infer performance profile:

$$pk = (s-1)\ell_{G_i} + \ell_{seed}$$
$$sig = \ell_{salt} + \ell_{seed\_tree} + w \cdot \ell_{mono} + \ell_{digest}$$

NIST Cat.	Parameter Set	Code Params n	k	q	Prot. Params t	w	s	pk (KiB)	sig (KiB)
1	LESS-1b	252	126	127	247	30	2	13.7	8.4
	LESS-1i				244	20	4	41.1	5.8
	LESS-1s				198	17	8	95.9	5.0
3	LESS-3b	400	200	127	759	33	2	34.5	16.8
	LESS-3s				895	26	3	68.9	13.4
5	LESS-5b	548	274	127	1352	40	2	64.6	29.8
	LESS-5s				907	37	3	129.0	26.6

Performance

Protocol parameters (t, w, s) infer performance profile:

$$pk = (s-1)\ell_{G_i} + \ell_{seed}$$
$$sig = \ell_{salt} + \ell_{seed\_tree} + w \cdot \ell_{mono} + \ell_{digest}$$

NIST Cat.	Parameter Set	Code Params n	k	q	Prot. Params t	w	s	pk (KiB)	sig (KiB)
1	LESS-1b	252	126	127	247	30	2	13.7	8.4
	LESS-1i				244	20	4	41.1	5.8
	LESS-1s				198	17	8	95.9	5.0
3	LESS-3b	400	200	127	759	33	2	34.5	16.8
	LESS-3s				895	26	3	68.9	13.4
5	LESS-5b	548	274	127	1352	40	2	64.6	29.8
	LESS-5s				907	37	3	129.0	26.6

Runtime is dominated by RREF computation, for both Sign and Verify.

LESS

LESS Keeps Getting...LESS!

LESS

↓

LESS

LESS Keeps Getting...LESS!

LESS

↓

LESS

↓

LESS

LESS

Barenghi, Biasse, P., Santini, *PQCrypto 2021*: original LESS-FM work with tweaks.

↓

LESS

↓

LESS

LESS

Barenghi, Biasse, P., Santini, *PQCrypto 2021*: original LESS-FM work with tweaks.

↓

LESS

P., Santini, *Asiacrypt 2023*: commit to information set to $\approx$ halve the signatures (in current spec).

↓

LESS

LESS Keeps Getting...LESS!

LESS

Barenghi, Biasse, P., Santini, *PQCrypto 2021*: original LESS-FM work with tweaks.

↓

LESS

P., Santini, *Asiacrypt 2023*: commit to information set to $\approx$ halve the signatures (in current spec).

↓

LESS

Chou, P., Santini, *preprint*: use canonical forms for compact representation (for next round).

Current parameters would change as follows.

Coming Up

Current parameters would change as follows.

NIST Cat.	Parameter Set	Code Params n	k	q	Prot. Params t	w	s	pk (KiB)	sig (KiB)	new sig (KiB)
1	LESS-1b	252	126	127	247	30	2	13.7	8.4	2.5
	LESS-1i				244	20	4	41.1	5.8	1.9
	LESS-1s				198	17	8	95.9	4.9	1.6
3	LESS-3b	400	200	127	759	33	2	34.5	16.5	5.3
	LESS-3s				895	26	3	68.9	13.4	4.6
5	LESS-5b	548	274	127	1352	40	2	64.6	29.2	7.8
	LESS-5s				907	37	3	129.0	26.5	6.8

Current parameters would change as follows.

NIST Cat.	Parameter Set	Code Params n	k	q	Prot. Params t	w	s	pk (KiB)	sig (KiB)	new sig (KiB)
1	LESS-1b	252	126	127	247	30	2	13.7	8.4	2.5
	LESS-1i				244	20	4	41.1	5.8	1.9
	LESS-1s				198	17	8	95.9	4.9	1.6
3	LESS-3b	400	200	127	759	33	2	34.5	16.5	5.3
	LESS-3s				895	26	3	68.9	13.4	4.6
5	LESS-5b	548	274	127	1352	40	2	64.6	29.2	7.8
	LESS-5s				907	37	3	129.0	26.5	6.8

These are among the smallest sizes seen so far.

More Current and Future Work

Full-fledged optimized implementation (AVX2), in progress.

More Current and Future Work

Full-fledged optimized implementation (AVX2), in progress.

High-performance hardware Implementation; first work, $\approx 2\times$ speed-up over AVX2.

(Beckwith, Wallace, Mohajerani, Gaj, 2023)

More Current and Future Work

Full-fledged optimized implementation (AVX2), in progress.

High-performance hardware Implementation; first work, $\approx 2\times$ speed-up over AVX2.
(Beckwith, Wallace, Mohajerani, Gaj, 2023)

Particularly suitable to develop protocols with advanced functionalities, e.g.:

More Current and Future Work

Full-fledged optimized implementation (AVX2), in progress.

High-performance hardware Implementation; first work, $\approx 2\times$ speed-up over AVX2.

(Beckwith, Wallace, Mohajerani, Gaj, 2023)

Particularly suitable to develop protocols with advanced functionalities, e.g.:

- Ring signatures.

  (Barenghi, Biasse, Ngo, P., Santini, 2022)

More Current and Future Work

Full-fledged optimized implementation (AVX2), in progress.

High-performance hardware Implementation; first work, $\approx 2\times$ speed-up over AVX2.
(Beckwith, Wallace, Mohajerani, Gaj, 2023)

Particularly suitable to develop protocols with advanced functionalities, e.g.:

- Ring signatures.
  (Barenghi, Biasse, Ngo, P., Santini, 2022)
- Threshold signatures.
  (Battagliola, Borin, Meneghetti, P., preprint)

More Current and Future Work

Full-fledged optimized implementation (AVX2), in progress.

High-performance hardware Implementation; first work, $\approx 2\times$ speed-up over AVX2.
(Beckwith, Wallace, Mohajerani, Gaj, 2023)

Particularly suitable to develop protocols with advanced functionalities, e.g.:

- Ring signatures.
  (Barenghi, Biasse, Ngo, P., Santini, 2022)
- Threshold signatures.
  (Battagliola, Borin, Meneghetti, P., preprint)
- Blind signatures.
  (Kuchta, LeGrow, P., preprint)

More Current and Future Work

Full-fledged optimized implementation (AVX2), in progress.

High-performance hardware Implementation; first work, $\approx 2\times$ speed-up over AVX2.
(Beckwith, Wallace, Mohajerani, Gaj, 2023)

Particularly suitable to develop protocols with advanced functionalities, e.g.:

- Ring signatures.
  (Barenghi, Biasse, Ngo, P., Santini, 2022)
- Threshold signatures.
  (Battagliola, Borin, Meneghetti, P., preprint)
- Blind signatures.
  (Kuchta, LeGrow, P., preprint)
- ...

More Current and Future Work

Full-fledged optimized implementation (AVX2), in progress.

High-performance hardware Implementation; first work, $\approx 2\times$ speed-up over AVX2.
(Beckwith, Wallace, Mohajerani, Gaj, 2023)

Particularly suitable to develop protocols with advanced functionalities, e.g.:

- Ring signatures.
  (Barenghi, Biasse, Ngo, P., Santini, 2022)
- Threshold signatures.
  (Battagliola, Borin, Meneghetti, P., preprint)
- Blind signatures.
  (Kuchta, LeGrow, P., preprint)
- ...

Stay tuned!

Thank you for listening!
Any questions?

<

https://www.less-project.com

References

J.-F. Biasse, G. Micheli, E. Persichetti, and P. Santini
LESS is More: Code-Based Signatures Without Syndromes.
AFRICACRYPT 2020.

A. Barenghi, J.-F. Biasse, E. Persichetti, and P. Santini
LESS-FM: Fine-Tuning Signatures from the Code Equivalence Problem.
PQCRYPTO 2021.

E. Petrank and M. R. Roth
Is code equivalence easy to decide?
IEEE Transactions on Information Theory, 43(5):1602–1604, 1997.

N. Sendrier
The Support Splitting Algorithm.
IEEE Transactions on Information Theory, 1193–1203, 2000.

M. A. Saeed-Taha
Algebraic Approach for Code Equivalence.
PhD Thesis.

M. Bardet and A. Otmani and M. A. Saeed-Taha
Permutation Code Equivalence is Not Harder Than Graph Isomorphism When Hulls Are Trivial.
IEEE ISIT 2019.

J. Leon
Computing automorphism groups of error-correcting codes.
IEEE Transactions on Information Theory, 28(3):496–511, 1982.

References

W. Beullens
Not Enough LESS: An Improved Algorithm for Solving Code Equivalence Problems over $\mathbb{F}_q$.
SAC 2020.

E. Persichetti, and P. Santini
A New Formulation of the Linear Equivalence Problem and Shorter LESS Signatures.
ASIACRYPT 2023.

T. Chou, E. Persichetti, and P. Santini
On Linear Equivalence, Canonical Forms, and Digital Signatures.
preprint, available at https://tungchou.github.io/papers/leq.pdf.

L. Beckwith, R. Wallace, K. Mohajerani and K. Gaj
A High-Performance Hardware Implementation of the LESS Digital Signature Scheme.
PQCRYPTO 2023.

A. Barenghi, J.-F. Biasse, T. Ngo, E. Persichetti, and P. Santini
Advanced Signature Functionalities from the Code Equivalence Problem.
International Journal of Computer Mathematics: Computer Systems Theory, 2022.

M. Battagliola, G. Borin, A. Meneghetti and E. Persichetti
Cutting the GRASS: Threshold GRoup Action Signature Schemes.
preprint, available at https://eprint.iacr.org/2023/859.