Analysis of the Multi-Key Optimization for SPECK

Marco Baldi[1], Michele Battagliola[3], Rahmi El Mechri[1,2], Paolo Santini[1], and Riccardo Schiavoni[1]

[1]Marche Polytechnic University, Ancona, Italy
[2] IMT School for Advanced Studies, Lucca, Italy
[3] St.Gallen University, St.Gallen, Switzerland

Abstract. SPECK has recently been introduced as a code-based digital signature scheme exploiting the similarities between the Permuted Kernel Problem (PKP) and the Permutation Code Equivalence Problem (PEP). By utilizing PEP as a trapdoor for PKP, SPECK achieves a significant speedup compared to LESS, replacing expensive Gaussian elimination with vector-matrix multiplications. This comes at the cost of larger signatures, since each round of the Zero Knowledge (ZK) protocol used in SPECK has soundness error 1/2.
A well known technique to reduce the soundness errors (and thus the number of parallel repetitions and, ultimately, the signature size) in ZK proofs based on group actions consists in giving multiple elements in the public keys. Despite being constructed around an instance of PEP, which admits a natural description as a code-based group action, the protocol employed in SPECK deviates from the traditional proof of knowledge used for group actions. As shown by the authors of SPECK, a naïve use of the multi-key optimization may lead to concrete security concerns. Therefore, the question on whether such a technique can be applied successfully to SPECK remains open.
In this paper, we give a positive answer to such a question. We revisit the multi-key optimization for SPECK and investigate the underlying security concerns in detail, identifying conditions and parameter constraints under which multiple keys can be securely integrated into the protocol. Even though, when instantiated with conservative parameters, the multi-key optimization leads to unappealing performance, we show there is wide room for improvements. Furthermore, we design a ring signature scheme based on the same underlying construction.

Keywords: Digital Signatures, Code-based cryptography, Code Equivalence, PKP, Ring Signatures

1 Introduction

In June 2023, NIST issued an additional call for post-quantum signature schemes, with the goal of either diversifying the mathematical landscape by moving away

from lattice-based cryptography or selecting algorithms with short signatures and fast verification.[1] The first round of the call ended in October 2024, when NIST selected the 14 candidate algorithms to advance to the next round [1] that ended in May 2026.

Among the 14 algorithms, LESS stands out for its short signatures. However, one of the reasons for its exclusion from the next round is that LESS suffers from extremely slow signature generation and verification due to the requirement of Gaussian elimination in both phases. Recently, SPECK was proposed [2] with the objective of eliminating Gaussian elimination, thus achieving faster signatures.

SPECK The main ingredient of SPECK is a new hardness assumption, called Permutation of Code and Kernel Problem (PECK). In a nutshell, instead of verifying the existence of a permutation map that transforms the first code into the second as in the case of the Permutation Code Equivalence (PEP), PECK verifies this relation only on some randomly sampled codewords.

Using this idea, they derive a simple ZK proof as follows: the secret key is a permutation matrix $\mathbf{P}$, while the public key is a pair of equivalent codes $\mathscr{C}, \mathscr{C}'$ represented by their generator matrices $\mathbf{G}, \mathbf{G}' = \mathsf{SF}(\mathbf{GP})$. The prover first samples a random permutation $\mathbf{P}^*$ and a random codeword $\mathbf{c} \in \mathscr{C}$ and then commits to codeword $\mathbf{c}' = \mathbf{cP}^*$. Later on, as responses, the prover either discloses the seed used to sample $\mathbf{P}^*$ and $\mathbf{c}$ or $\mathbf{P}' = \mathbf{P}^{*^{-1}}\mathbf{P}$: in the first case, the verifier just repeats the same operations carried out by the prover and checks the correctness of the commitment, while in the second case they check that $\mathbf{c}'\mathbf{P}'\mathbf{H}'^{\top} = \mathbf{0}$, where $\mathbf{H}'$ is the parity check matrix corresponding to $\mathbf{G}'$.

Interestingly, SPECK does not use the multi-key optimization. Instead, the authors explicitly reject it due to security concerns. Indeed, a crucial fact for SPECK security is that the codeword $\mathbf{c}$ is chosen randomly and not "carefully selected" by the prover. As explained before, this fact is checked only in one of the two cases, when the prover reveals the seed used. Following this observation, an attacker could find a codeword that lies in the intersection of two (or more) public keys, effectively being able to respond to more than one challenge. This attack can be further refined when considering the intersection of the permutations of public keys. This completely nullifies the knowledge error reduction, making the use of multiple public keys useless (or even dangerous) for SPECK.

1.1 Our contribution

In this paper, we analyze the problem of, given a pair of permutationally equivalent codes $\mathscr{C}, \mathscr{C}'$, finding codewords $\mathbf{c}, \mathbf{c}'$ that are permutations of each other such that $\mathbf{c} \in \mathscr{C}$ and $\mathbf{c}' \in \mathscr{C}'$. Notice that this problem is different from PECK, since $\mathbf{c}$ is not randomly sampled but can be freely chosen. As noted before, this problem is at the basis of the security concerns regarding the multi-key optimization for SPECK.

[1] https://csrc.nist.gov/csrc/media/Projects/pqc-dig-sig/documents/call-for-proposals-dig-sig-sept-2022.pdf

Our main result is finding conditions on $\mathscr{C}$ and $\mathbf{c}$ so that the above problem is hard. We corroborate our theoretical analysis of the problem with several numerical simulations that try to estimate its complexity.

As a consequence, we change SPECK parameters to be suitable for the multi-key optimization and derive an optimized implementation for it. Unfortunately, when sticking to our security estimations, our implementation falls short of improving SPECK performances. However, when relaxing the constraints, we were able to almost halve the signing and verification time, at the price of bigger keys and slightly longer signatures.

Finally, as a second result, we also notice that PECK can also be used to design a ring signature based on SPECK. While lacking an implementation, based on the performance of the original SPECK signature, we expect our signature to maintain the same computational speed-up when compared to [6], at the price of a slightly bigger signature.

1.2 Paper organization

In Section 2 we introduce the notation used in this work and the preliminaries needed. In Section 3 we describe the issues related to the multi-key optimization for SPECK and analyze PECK in relation to it, showing how it can be adapted to be secure also in the case of multiple public keys. In Section 4 we show the obtained protocol, discussing its performance, while in Section 5 we show a ring signature derived from it.

2 Preliminaries

In this section, we establish the notation we use throughout the paper and recall the necessary background concepts regarding linear codes and SPECK signature scheme.

2.1 Notation

We denote by $\mathbb{F}_q$ the finite field with q elements, where q can be any prime power. We will use boldface uppercase letters to denote matrices, and boldface lowercase letters to denote vectors. The identity matrix of size k is denoted by $\mathbf{I}_k$. The null vector or matrix is denoted by $\mathbf{0}$; the dimensions of $\mathbf{0}$ will always be clear from the context.

We denote by S_n the symmetric group on n elements and view its elements as permutations. We usually represent permutations through the associated $n \times n$ matrices and, with some abuse of notation, will write $\mathbf{P} \in S_n$ to specify that $\mathbf{P}$ is a permutation matrix. The group of non-singular square matrices with dimension k and values over $\mathbb{F}_q$ is denoted by $\mathrm{GL}_k(\mathbb{F}_q)$.

Cryptographic notation As usual, λ indicates the security parameter. We use $\mathsf{Hash} : \{0,1\}^* \to \{0,1\}^{2\lambda}$ to indicate a cryptographic hash function. We use the symbol $\xleftarrow{\$}$ to denote uniform sampling from a set. For a given distribution $\mathcal{D}$, we write $x \leftarrow \mathcal{D}$ if x is sampled from $\mathcal{D}$. Finally, if the seed $\mathtt{Seed}$ used for sampling is relevant we write $\xleftarrow{\mathtt{Seed}}$.

When writing pseudocode we reserve the symbol = for equality testing, while we use := to denote assignment (e.g. $x := a + b$ denotes that the variable x assumes the value $a + b$).

We refer to [10] for the basic notions about ZK proofs and the Fiat-Shamir Transform.

2.2 Linear Codes

We now provide some basic notions on coding theory, for further details we refer to [15].

A $[n, k]$ *linear code* (from now on simply code) $\mathscr{C}$ is a k-dimensional subspace of $\mathbb{F}_q^n$. The values n and k are respectively the *length* and the *dimension* of the code. An element $\mathbf{c} \in \mathscr{C}$ is called a codeword. We define *code rate*, or simply, rate the quantity $R = \frac{k}{n}$. We represent a code $\mathscr{C} \subseteq \mathbb{F}_q^n$ with dimension k either using its *generator matrix*, that is, a full-rank $\mathbf{G} \in \mathbb{F}_q^{k \times n}$ such that the rows form a basis for the code or using its *parity-check matrix*, a full rank $\mathbf{H} \in \mathbb{F}_q^{(n-k) \times n}$ whose rows form a basis for the null space of the code.

Clearly, for a generator matrix $\mathbf{G}$ and a parity-check matrix $\mathbf{H}$ for the same code, we have that $\mathbf{G}\mathbf{H}^\top = \mathbf{0}$.

We say that a generator matrix $\mathbf{G}$ is in *systematic form* if $\mathbf{G} = (\mathbf{I}_k, \mathbf{A})$ with $\mathbf{A} \in \mathbb{F}_q^{k \times (n-k)}$, from which we obtain the systematic parity-check matrix as $(\mathbf{B}, \mathbf{I}_{n-k})$, with $\mathbf{B} = -\mathbf{A}^\top$. We indicate as SF the function that, on input a generator matrix, returns its systematic form (when it can indeed be defined).

Notice that the systematic form of a generator matrix $\mathbf{G}$ may not exist: this happens whenever the first k columns form a singular matrix. In this case, we generalize the notion of systematic form through that of "standard" form, where the $k \times k$ identity matrix shows up somewhere among the columns of $\mathbf{G}$ (the same holds also for parity-check matrices). Without loss of generality, we assume that all the matrices admit a systematic form.

The hull of a linear code The hull of a code is the subspace given by the intersection between the code and its dual. Formally:

Definition 1 (Hull). *Given a linear $[n, k]$-code $\mathscr{C}$ the* hull *of $\mathscr{C}$ is defined as:*

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

We say that the hull of a code is trivial if it contains only the null vector.

To avoid confusion, in this paper we use the same terminology of [2]. In particular, we call *weakly self-dual* those codes having hull dimension $h \geqslant 1$.

When $h = k$, we say that the code is *self-orthogonal.* As we will explain later, low hull dimension makes solving PEP easy [4, 20], for this reason throughout this paper we only consider codes having hull dimension high enough, typically self-orthogonal codes. [2]

2.3 PEP and PKP

In this paper, we consider the Permutation Equivalence Problem (PEP) which, given two linear codes, asks to find a permutation mapping one code onto the other. Formally, the problem is as follows.

Problem 1 (Permutation Equivalence Problem (PEP)) *Given two linear $[n, k]$-codes $\mathscr{C}, \mathscr{C}' \subseteq \mathbb{F}_q^n$, with respective generator matrices $\mathbf{G}, \mathbf{G}' \in \mathbb{F}_q^{k \times n}$, find (if it exists) a permutation $\mathbf{P} \in S_n$ and a non-singular $\mathbf{S} \in GL_k(\mathbb{F}_q)$ such that $\mathbf{G}' = \mathbf{SGP}$.*

Notice that the matrix $\mathbf{S}$ is not part of the witness. Indeed, one can simply check that $\mathbf{GPH}'^{\top} = \mathbf{0}$, with $\mathbf{H}'$ being a parity-check matrix for $\mathscr{C}'$.

We also recall the Permuted Kernel Problem (PKP). Since one of our goals is to analyze the relation between PEP and PKP we state it in terms of coding theory.

Problem 2 (Permuted Kernel Problem (PKP)) *Given a linear $[n, k]$-code $\mathscr{C} \subseteq \mathbb{F}_q^n$ with parity-check matrix $\mathbf{H} \in \mathbb{F}_q^{(n-k) \times n}$ and a vector $\mathbf{c} \in \mathbb{F}_q^n$, find (if it exists) a permutation $\mathbf{P} \in S_n$ such that $\mathbf{cPH}^{\top} = \mathbf{0}$.*

We summarize here some general results about the hardness of PEP and PKP. For further details we refer to [5, 19].

- Attacks based on the hull [4, 20], that have complexity exponential in the hull dimension. Thus PEP is easy when the hull is small.
- Attacks based on low weight codewords [8, 17] and the meet-in-the-middle based on canonical forms [13]. These kinds of attacks on PEP take time which is exponential in the code length.
- Attacks based on power-codes [7] and [18, Appendix B]. These attacks work only on codes having extremely low rate or hull (e.g. $k < \sqrt{2n}$ or $h < \sqrt{2n}$).
- Most of the attacks on PKP require to brute-force over all the possible permutations of $\mathbf{c}$ using a meet-in-the-middle approach [16]. Possible speed-ups includes techniques such as applying a filtering step exploiting the parity-check equations [19] or recursion [12]. Interestingly, while PKP has usually been studied in the scenario in which the vector $\mathbf{c}$ has all different entries (that often requires working on a finite field $\mathbb{F}_q$ with q big), the SPECK protocol [2] works on fields having low q and the vectors have repeated entries.

[2] Notice that SPECK uses self-dual codes, that are self-orthogonal codes having rate 1/2. In this paper, we reduce the rate, so we do not use the same kind of codes.

2.4 PECK

Introduced in [2], PECK was designed as a relaxed version of PKP and PEP. Roughly speaking, the idea is that instead of asking for a permutation that maps two codes (as it is the case for PEP), we ask for it only on random codewords. Formally, it can be defined as follows:

Problem 3 (Permutation Equivalence of Codes and Kernel (PECK)) *Let $\mathscr{C}, \mathscr{C}'$ be two linear $[n,k]$-codes over $\mathbb{F}_q$ such that $\mathscr{C}' = \pi(\mathscr{C})$ for $\pi \in S_n$. Given $\mathbf{c} \in \mathscr{C}$, find $\tilde{\pi}$ such that $\tilde{\pi}(\mathbf{c}) \in \mathscr{C}'$.*

The connection with PKP and PEP is quite clear: on one hand, it is easier than PKP since the codeword $\mathbf{c}$ is sampled from an equivalent code $\mathscr{C}'$ which is given as an additional input; on the other hand any solver of PEP can simply recover the permutation between $\mathscr{C}$ and $\mathscr{C}'$ and thus can solve PECK.

In [2] the complexity class of PECK has been analyzed by showing its relations with PEP and PKP, as q and n vary. In particular, the authors showed that for small q the problem is easy while for large q it is as hard as PEP. However, the authors ultimately resort to using only instances having relatively small q (in particular for $q \approx \frac{n}{2}$) for efficiency reasons. This choice of q was also one of the reasons why the authors ruled out the multi-key optimization. Indeed, for PECK instances over small fields, the existence of multiple valid solutions $\tilde{\pi}$ precludes the reduction of the soundness error via the multi-key technique, as we will see later.

2.5 SPECK signature scheme

Intuitively, SPECK [2] is based on the permutation version of LESS. The main difference is that instead of committing to an entire permuted code, in SPECK the prover commits to a randomly permuted codeword. In this sense, the idea is that it is enough to show the knowledge of a permutation that maps random codewords from one code into the other to prove the knowledge of the permutation that maps the two codes as a whole.

SPECK public key is given by a generator matrix $\mathbf{G} \in \mathbb{F}_q^{k \times n}$ for $\mathscr{C}$ and a parity-check matrix $\mathbf{H}' \in \mathbb{F}_q^{(n-k) \times n}$ for $\mathscr{C}'$, so that $\mathbf{G}\mathbf{P}\mathbf{H}'^{\top} = \mathbf{0}$ for a permutation $\mathbf{P} \in S_n$ which is the secret key. First, the prover samples a random ephemeral permutation $\mathbf{P}^* \xleftarrow{\$} S_n$ and a random vector $\mathbf{u} \xleftarrow{\$} \mathbb{F}_q^k$ and then computes the codeword $\mathbf{c} = \mathbf{u}\mathbf{G}\mathbf{P}^*$. When the challenge is 0, the prover responds with the seed used to sample $(\mathbf{u}, \mathbf{P}^*)$, so that the verifier can check that indeed the commitment was honestly generated and that both $\mathbf{P}^*$ and $\mathbf{u}$ were indeed randomly sampled. When the challenge is 1, the prover responds instead with $\mathbf{P}' = \mathbf{P}^{*-1}\mathbf{P}$, so that the verifier can check that indeed $\mathbf{c}\mathbf{P}'\mathbf{H}'^{\top} = \mathbf{0}$.

2.6 Multi-key optimization

Introduced in [14], the multi-key optimization is a widely used technique for identification schemes based on group actions with a binary challenge space. The

main goal is to achieve shorter and faster signatures by reducing the soundness error (and thus reducing the number of parallel repetitions) at the expense of larger public keys. In a nutshell, rather than utilizing a single key-pair $(\mathsf{sk}, \mathsf{pk})$, the prover generates s distinct key-pairs $(\mathsf{sk}_i, \mathsf{pk}_i)$, $i = 1, ..., s$. During the signing algorithm, challenges are sampled from the set $\{0, \ldots, s\}$ instead of the standard $\{0, 1\}$, so that each of them selects one of the public keys, for which a response is produced using the corresponding private key. In particular, the response for a challenge $\mathtt{ch} = 0$ remains unchanged, while the response for a challenge $\mathtt{ch} = i^*$ corresponds to the response for $\mathtt{ch} = 1$ of the base protocol executed using the key-pair $(\mathsf{sk}_{i^*}, \mathsf{pk}_{i^*})$. This approach reduces the soundness error from $\frac{1}{2}$ to $\frac{1}{s+1}$, allowing for a reduction in the number of parallel repetitions by a factor of $\log_2(s)$ compared to the base protocol.

In the next section, we explain in detail the multi-key optimization for SPECK and the issues related to it.

3 SPECK multi-key

In this section, we will focus on the multi-key optimization of SPECK. We first describe briefly why the original SPECK protocol is incompatible with this optimization, then we analyze PECK more in detail and slightly modify it to solve the issues. Finally, we discuss the security of our proposed solutions.

3.1 Main problem

We now show a naïve application of the multi-key technique explained in Section 2.6 for SPECK. Let $\mathscr{C}_0, \mathscr{C}_1, \ldots, \mathscr{C}_s \subseteq \mathbb{F}_q^n$ be s k-dimensional linear codes having generator matrices $\mathbf{G}_0, ..., \mathbf{G}_s$ such that:

$$\mathscr{C}_i = \pi_i(\mathscr{C}_0), \quad \pi_i \in S_n, \text{ with } 1 \leqslant i \leqslant s.$$

As in the base protocol, the prover first commits to a permutation of a random codeword of $\mathscr{C}_0$, i.e., $\mathbf{c} = \pi^*(\mathbf{uG})$ for randomly sampled $\pi^* \in S_n$ and $\mathbf{u} \in \mathbb{F}_q^k$. Then, according to the challenge $\mathtt{ch} \in \{0, ..., s\}$ the prover does:

- if $\mathtt{ch} = 0$ then the prover reveals the seed used to sample π^* and $\mathbf{u}$,
- if $\mathtt{ch} \in \{1, \ldots, s\}$, the prover sends $(\mathbf{c}, \pi_{\mathtt{ch}} \circ \pi^{*-1})$.

Finally, the verifier checks that the commitment is honestly generated (when $\mathtt{ch} = 0$) or that $\pi_{\mathtt{ch}} \circ \pi^{*-1}(\mathbf{c}) \in \mathscr{C}_{\mathtt{ch}}$ (when $\mathtt{ch} \neq 0$).

Normally, this construction would achieve a soundness error equal to $\frac{1}{(s+1)}$. However, this requires that finding a codeword $\mathbf{c}'$ so that it is easy to find multiple permutations that map it into multiple public keys is hard. Unfortunately, this is only the case when for $i \neq j \in \{0, 1, \ldots, s\}$ and for all $\mathbf{c} \in \mathscr{C}_i$ the only permutation $\tilde{\pi}$ that maps $\mathbf{c}$ into $\mathscr{C}_j$ is $\pi_i^{-1} \circ \pi_j$, i.e., the one such that $\mathscr{C}_j = (\pi_i^{-1} \circ \pi_j)(\mathscr{C}_i)$.

Given a random instance of PECK $(\mathscr{C}, \mathscr{C}' = \pi(\mathscr{C}), \mathbf{c})$, we want to estimate the probability that spurious solutions exist, namely that there is a permutation $\tilde{\pi} \in S_n \backslash \{\pi\}$ such that $\tilde{\pi}(\mathbf{c}) \in \mathscr{C}'$.

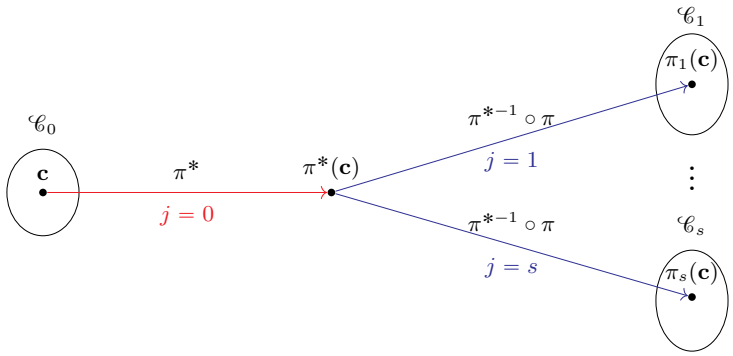

Fig. 1: Schematic view of SPECK with multiple public keys.

If this probability is not negligible, it means that it is possible given a pair $\mathscr{C}_i = \pi_i(\mathscr{C}_0), \mathscr{C}_j = \pi_j(\mathscr{C}_0)$ to find one permutation $\tilde{\pi} \neq \pi_i^{-1} \circ \pi_j$ such that

$$\mathscr{C}_{ij} = \tilde{\pi}(\mathscr{C}_i) \cap \mathscr{C}_j \neq \{\mathbf{0}\} \tag{1}$$

This allows a malicious prover $\mathcal{A}$ to respond correctly when $\mathtt{ch} = j$ and $\mathtt{ch} = i$ as follows:

- $\mathcal{A}$ prepares the commitment as $\mathbf{c} = \pi^*(\mathbf{c})$ for $(\mathbf{c}, \pi^*) \xleftarrow{\$} \mathscr{C}_{ij} \times S_n$
- For $\mathtt{ch} = j$ it responds with π^{*-1}, showing that $\pi^{*-1}(\mathbf{c}^*) \in \mathscr{C}_j$.
- For $\mathtt{ch} = i$ it responds with $\pi^{*-1} \circ \tilde{\pi}^{-1}$, showing that $(\pi^{*-1} \circ \tilde{\pi}^{-1})(\mathbf{c}^*) \in \mathscr{C}_i$.

This attack can be extended to consider every pair of codes $\mathscr{C}_i, \mathscr{C}_j$. It follows that $\mathcal{A}$ would be able to prepare a commitment for which it is able to respond correctly for any $\mathtt{ch} \neq 0$ and, consequently, the soundness error is not $\frac{1}{s+1}$ but is increased up to $\frac{1}{2}$. [3]

3.2 Instances of PECK with unique solution

In light of the previous section, we want to deeply investigate the relation between PECK and PEP. Let $(\mathscr{C}, \mathscr{C}')$ be a random instance of PEP, it is quite intuitive that we can associate with this instance as many instances of PECK as there are codewords of $\mathscr{C}$ (or $\mathscr{C}'$, equivalently). More formally we can define the set of PECK associated to a random PEP as:

$$\mathcal{D}_{PECK}(\mathscr{C}, \mathscr{C}') = \{(\mathscr{C}, \mathscr{C}', \mathbf{c}) \,|\, \mathbf{c} \in \mathscr{C}\}$$

[3] It is worth noticing that while we did not manage to prove the security without this assumption, we did not manage to find any practical attack that exploits the previous fact. Indeed, we conjecture that even if multiple permutation exist, finding $\mathscr{C}_{ij}$ of Equation (1) is not easy as long as some pathological solutions are excluded, such as the null codeword.

What we want to do is study the range of parameters q, n, R for which it is possible to define a particular subset of instances within $\mathcal{D}_{PECK}$ that are as hard as the associated PEP instance, that is, those for which the only permutation π that maps $\mathbf{c}$ into $\mathscr{C}'$ is the same one that maps $\mathscr{C}$ to $\mathscr{C}'$.

First of all, it is easy to see that a necessary condition for having a uniqueness of solutions is that $\mathbf{c}$ has no repeated entries; then by denoting with $\mathscr{C}_{\mathsf{diff}}$ the set of codewords of $\mathscr{C}$ having all different entries we can define the following:

$$D^H_{PECK}(\mathscr{C}, \mathscr{C}') := \{(\mathscr{C}, \mathscr{C}', \mathbf{c}) \,|\, \mathbf{c} \in \mathscr{C}_{\mathsf{diff}}\} \subseteq D_{PECK}(\mathscr{C}, \mathscr{C}')$$

To analyze the hardness of the PECK instances within this subset compared to the corresponding $\mathsf{PEP}_{q,n,k}$ one can proceed as follows. Throughout this section, we model a $[n, k]$-code $\mathscr{C}$ as set of q^k random vectors of length n, instead of a vector space. Indeed, when taking into consideration the linearity, the computations becomes very hard and for our purpose this approximation is enough, as shown later by our simulations.

Under this consideration, for a given $\pi \in S_n$ we have that for each $\mathbf{c} \in \mathscr{C}, \pi' \neq \pi$ the event $\{\pi'(\mathbf{c}) \in \pi(\mathscr{C}')\}$ happens with probability:

$$\Pr\big(\pi'(\mathbf{c}) \in \mathscr{C}'\big) \approx \frac{|\mathscr{C}'|}{q^n} = q^{k-n}.$$

The number Z of permutations $\pi' \neq \pi$ mapping $\mathbf{c}$ into $\mathscr{C}'$ can be modeled as a binomial random variable with $n! - 1$ trials and success probability q^{k-n}.

Therefore, for any integer $0 \leqslant z \leqslant n! - 1$, we have

$$\Pr(Z = z) \approx \binom{n! - 1}{z} (q^{k-n})^z (1 - q^{k-n})^{n!-1-z}. \tag{2}$$

We are interested in the probability that at least one spurious permutation occurs; using Markov's inequality this probability can be lower bounded as follows:

$$p_{q,n,k} = \Pr(Z \geqslant 1) \leqslant \mathbb{E}[Z] = (n! - 1) q^{k-n} \tag{3}$$

Using Stirling's upper bound, $n! \leqslant \sqrt{2\pi n} \left(\frac{n}{e}\right)^n e^{\frac{1}{12n}}$, this probability can be further upper bounded

$$p_{q,k,n} \leqslant \frac{n! - 1}{q^{n-k}} < \frac{n!}{q^{n-k}} \leqslant \frac{\sqrt{2\pi n} \left(\frac{n}{e}\right)^n e^{\frac{1}{12n}}}{q^{n(1-R)}} \leqslant \sqrt{2\pi n} \cdot e^{n(\ln(n) - (1-R)\ln(q) - 1) + \frac{1}{12n}}$$

where $R = \frac{k}{n}$ is the code rate.

For large n it holds that $e^n \gg \sqrt{2\pi n}$, then by setting $\alpha = (\ln(n) - (1 - R)\ln(q) - 1)$, we have that for large q, n the value $\sqrt{2\pi n} \cdot e^{\alpha n \frac{1}{12n}} \sim e^{\alpha n}$ goes to zero as long as $\alpha < 0$.

Then a random instance $\{\mathbf{c}, \mathscr{C}, \mathscr{C}'\} \leftarrow D^H_{PECK}(\mathscr{C}, \mathscr{C}')$ has unique solution when the following condition holds:

$$p_{q,k,n} \approx 0 \Leftrightarrow q > \left(\frac{n}{e}\right)^{\frac{1}{1-R}} \tag{4}$$

The next step is to define the conditions so that the number of codewords $\mathbf{c}$ for which there exists at least one other permutation $\tilde{\pi} \neq \pi$ is negligible. We can approximate this quantity as

$$|\mathscr{C}_{\mathsf{diff}}| \frac{n!-1}{q^{n-k}} (q-1) = \frac{\binom{q}{n} n!}{q^n} q^k \frac{n!-1}{q^{n-k}} (q-1) \tag{5}$$

Where we introduced the factor $q-1$ to partially account for the linearity: indeed if there exist a codeword $\mathbf{c} \in \mathscr{C}$ that is mapped into $\pi(\mathscr{C})$ by different permutations $\tilde{\pi} \neq \pi$ then there exist $q-1$ of them, namely all its multiples $a\mathbf{c}$ for $1 < a < q-1$. Knowing that $k = Rn$ this formula can be bounded as follows:

$$|\mathscr{C}_{\mathsf{diff}}| \frac{n!-1}{q^{n-k}} (q-1) \leqslant \frac{q!}{(q-n)!} \frac{n!}{q^{2(n-k)-1}} = \left(\prod_{i=0}^{n} \frac{q-i}{q} \right) \frac{n!}{q^{n(1-2R)-1}}.$$

The correction factor $q-1$ explains that the probability that a single codeword $\mathbf{c} \in \mathscr{C}$ is mapped to a code $\pi(\mathscr{C})$ by other permutations $\tilde{\pi} \neq \pi$ is not independent; in fact, if this happens for $\mathbf{c}$, it also happens for $\alpha\mathbf{c}$ for $1 < \alpha < q-1$. Using Stirling approximation this become:

$$\left(\prod_{i=0}^{n} \frac{q-i}{q} \right) \frac{\sqrt{2\pi n}(\frac{n}{e})^n}{q^{n(1-2R)-1}} = \left(\prod_{i=0}^{n} \frac{q-i}{q} \right) \frac{(\sqrt{2\pi n})}{e} e^{n(\ln n - (1-2R)\ln(q) - 1)}$$

As before, for large n it holds that $e^n \gg \frac{\sqrt{2\pi n}}{e}$. Thus, knowing that $0 \leqslant \prod_{i=0}^{n} \frac{q-i}{q} \leqslant 1$, to obtain that this formula is smaller than 1 we need $\ln n - (1-2R)\ln(q) - 1 \leqslant 0$ and this happens when $q > (\frac{n}{e})^{\frac{1}{1-2R}}$. We can derive the following condition on q:

$$|\mathscr{C}_{\mathsf{diff}}| \frac{n!-1}{q^{n-k}} \leqslant 1 \Leftrightarrow q > \left(\frac{n}{e}\right)^{\frac{1}{1-2R}} \tag{6}$$

On the other hand, when q is not small, the formula becomes

$$\begin{aligned} |\mathscr{C}_{\mathsf{diff}}| \frac{n!-1}{q^{n-k}} \leqslant 1 \Leftrightarrow \ln n - (1-2R)\ln q - 1 < \ln\left(\frac{e}{\sqrt{2\pi n}}\right) \frac{1}{n} \Leftrightarrow \\ \Leftrightarrow \ln q > \frac{(\ln n - 1 - \ln\left(\frac{e}{\sqrt{2\pi n}}\right)\frac{1}{n})}{1-2R} \Leftrightarrow q > \left(\frac{n}{e} \left(\frac{\sqrt{2\pi n}}{e} \right)^{\frac{1}{n}} \right)^{\frac{1}{1-2R}} \end{aligned} \tag{7}$$

Under this condition we have that PECK is equivalent to PEP. More formally we have the following

Problem 4 (PECK^H) *Let $\mathscr{C}, \mathscr{C}'$ be two linear $[n,k]$-codes over $\mathbb{F}_q$ such that $\mathscr{C}' = \pi(\mathscr{C})$ for $\pi \in S_n$. Given $\mathbf{c} \in \mathscr{C}_{\mathsf{diff}}$, find $\tilde{\pi}$ such that $\tilde{\pi}(\mathbf{c}) \in \mathscr{C}'$.*

Theorem 1. *Let n, k and q be as Equation* (6)*, then, on average,* PEP *is equivalent to* PECK^H*.*

Proof. We only need to show one of the two directions, since the other is trivial.

Let $\mathcal{S}_{\mathsf{PECK}}$ be a solver for PECK^H, we show it can be used to define a solver $\mathcal{S}_{\mathsf{PEP}}$ for PEP.

- Let $\mathscr{C}, \mathscr{C}'$ be an instance of PEP. $\mathcal{S}_{\mathsf{PECK}}$ picks a random $\mathbf{c} \in \mathscr{C}$ having no repeated entries.
- $\mathcal{S}_{\mathsf{PEP}}$ starts $\mathcal{S}_{\mathsf{PECK}}$ on input $(\mathscr{C}, \mathscr{C}', \mathbf{c})$ and receive back a permutation π' so that $\pi'(\mathbf{c}) \in \mathbf{c}'$ with non-negligible probability ϵ.
- By hypothesis there exist π so that $\pi(\mathscr{C}) = \mathscr{C}'$, so clearly $\pi(\mathbf{c}) \in \mathscr{C}'$. Since n, k and q satisfies Equation (6), on average there is a unique solution to PECK^H, thus $\pi = \pi'$, thus $\mathcal{S}_{\mathsf{PEP}}$ solves PEP with the same non-negligible probability ϵ.

3.3 Testing our estimation

To validate our theoretical estimation we show the empirical results of an exhaustive search for pairs of codewords having the same multi-set (i.e. one is the permutation of the other) among all the vectors of a self-orthogonal code having all distinct entries . For efficiency reasons we are able to perform this test only when n is small, and thus we check Equation (7).

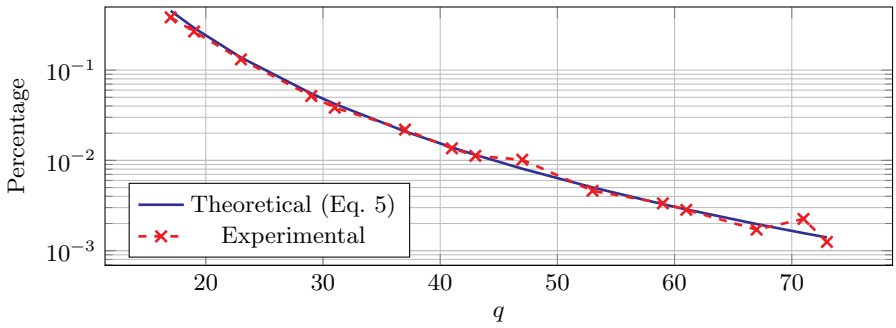

Fig. 2: Percentage of codewords with all different entries having the same multi-set, with $n = 8, k = 3$, for growing q. The experimental results are the average of 100 trials for each q.

Figure 2 shows the experimental results with $n = 8$ and $k = 3$. For growing q, our theoretical estimation in Equation (5) and our experimental results become very close. In particular, according to Equation (7), with $q > 200$ we have a negligible probability of having multiple codewords with the same multi-set. As the data shows our estimations are quite conservative and the percentage is less than 1 even with $q \simeq 70$.

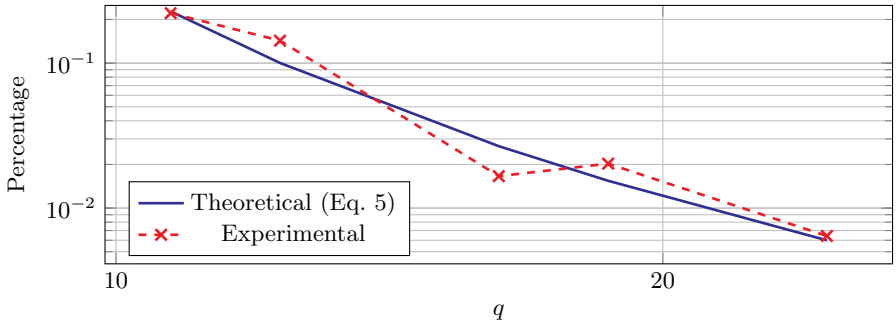

Fig. 3: Percentage of codewords with all different entries having the same multi-set, with $n = 8, k = 2$, for growing q. The experimental results are the average of 100 trials for each q.

Figure 3 shows the experimental results with $n = 8$ and $k = 2$. For growing q, our theoretical estimation in Equation (5) and our experimental results become very close. In particular, according to Equation (7), with $q > 15$ we have a negligible probability of having multiple codewords with the same multi-set. Also in this case our theoretical analysis matches closely the experimental results.

4 Protocol

We describe here our protocol. Since the protocol is a straightforward adaptation of the original SPECK protocol using the multi-key optimization we simply report the high-level description of the interactive version. We report the signature obtained by the application of the Fiat-Shamir Transform in Section A, where we also include the various optimizations used in the original work.

High-level description The public key is given by a generator matrix $\mathbf{G} \in F_q^{k \times n}$ for $\mathscr{C}$ and s parity-check matrices $\mathbf{H}_1, ..., \mathbf{H}_s \in \mathbb{F}_q^{(n-k) \times n}$ for $\mathscr{C}_1, ..., \mathscr{C}_s$, so that for all $i = 1, ..., s$ we have $\mathbf{G}\mathbf{P}_i\mathbf{H}_i'^{\top} = \mathbf{0}$ for a permutation $\mathbf{P}_i \in S_n$. The set of all $\mathbf{P}_i$ is the secret key. The prover first samples a random permutation $\mathbf{P}^* \xleftarrow{\$} S_n$ and a random vector $\mathbf{u} \xleftarrow{\$} \mathbb{F}_q^k$ and then computes the commitment as $\mathbf{c} = \mathbf{u}\mathbf{G}\mathbf{P}^*$. When the challenge is 0, the prover responds with $(\mathbf{u}, \mathbf{P}^*)$, so that the verifier can check that indeed the commitment was honestly generated. When the challenge is i with $1 \leqslant i \leqslant s$, the prover responds instead with $\mathbf{P}' = \mathbf{P}^{*-1}\mathbf{P}_i$, so that the verifier can check that indeed $\mathbf{c}\mathbf{P}'\mathbf{H}_i'^{\top} = \mathbf{0}$.

Proposition 1. *Let n, k and q be as Equation* (6)*. Then* SPECK *with s public key has soundness $\frac{1}{s+1}$ and is Honest-Verifier Zero-Knowledge.*

Proof (Sketch). The proof follows trivially from well known fact about the multi-key optimization and the fact that for all $i, j \in \{1, \ldots, s\}$ it is impossible to find $c \in \mathcal{C}_{ij}$ (as shown in Section 3).

The Honest-Verifier Zero-Knowledge follows trivially from the security of the original protocol: given a challenge $\mathtt{ch}$ the simulator can simply sample the response for the code $\mathscr{C}_{\mathtt{ch}}$ (or $\mathscr{C}$ when $\mathtt{ch} = 0$).

In Section A, we include the pseudocode for the signature algorithm, obtained by applying the Fiat-Shamir transform to the previously explained interactive protocol.

4.1 Parameters

Now, we will explain the rationale behind the choice of the concrete parameters, based on security and performance criteria. In Section 3 we characterized the instances of PECK with unique solution. In particular, we have formalized a specific condition, namely Equation (6), such that PECK is as hard as PEP. Given q and R, the maximum value of n that satisfies Equation (6) is given by $n = e \cdot q^{1-2R}$; it is easy to see that the closer R is to 0.5, the less n increases as q increases. This implies that for high rates, there are no configurations of parameters that simultaneously satisfy Equation (6) and are secure, since n would be too low.

Parameters satisfying Equation (6)					
q	n	R	CF_{MITM}	LB_{CS}	
8191	400	0.2	171	149	
16381	400	0.2	171	149	
16381	320	0.25	156	135	
Parameters satisfying only Equation (4)					
q	n	R	CF_{MITM}	LB_{CS}	KMP
8191	320	0.25	156	135	481.5
8191	274	0.35	155	133	546.6
8191	248	0.45	150	129	615.8
16381	274	0.35	155	133	531

Table 1: Parameters hardness according to PEP and PKP solvers.

The first set of parameters is the one for which Equation (6) is satisfied, and according to Equation (5), the instances they describe are as hard as PEP; this consideration allows us to evaluate their security level considering only PEP solvers. In particular, we will consider the theoretical lower bound for algorithms

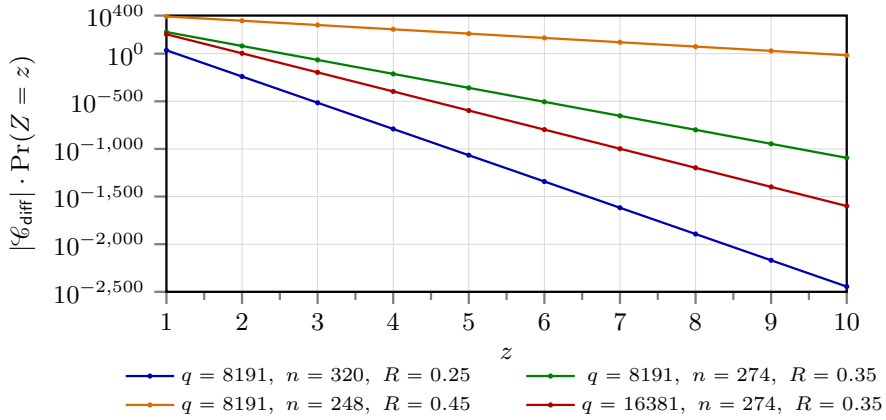

Fig. 4: Average number of codewords $\mathbf{c} \in \mathscr{C}$ with z distinct permutations π' such that $\pi'(\mathbf{c}) \neq \pi(\mathscr{C})$, for various values of z and field size q.

in the code-word search framework and the meet-in-the-middle algorithm based on the canonical form described in [13].

The other set satisfies only Equation (4), this means that, on average, a random instance has a unique solution, but this does not exclude that there are instances with spurious solutions. When Equation (7) is not met we also need to evaluate the performances of the solvers for PKP which can be applied, as shown in [2], to PECK. Among these algorithms, we will consider only the KMP algorithm [16], which, for large values of q, has been shown to be the best solver for the PKP problem.

However, as shown in Figure 4, as the number z of spurious solutions increases, the average number of codewords with z spurious solutions tends to zero; this, combined with the fact that there is no efficient way to identify a pair of codewords with the same multiset within a code, allows us to limit our security analysis to solvers for PEP and PKP.

4.2 Performance analysis

Table 2 shows the performance of our signature scheme, leveraging the AVX2 instruction set. Benchmarks were performed on an AMD Ryzen 7 PRO 8840U CPU clocked at 3.3GHz with Precision Boost disabled. The main reason for such a drastic difference between $n = 400$ and $n = 248$ is the number of samples needed to find a codeword with distinct entries. Indeed, when k is small, it requires several samplings (see Section C). On the other hand, increasing k allows one to significantly reduce the number of samples needed, allowing for a much faster signature process. This is also the reason why the verification algorithm is much faster, since the verifier needs to perform the sampling only when $\mathtt{ch} = 0$. See Section C for more details about the sampling algorithm. This

q	n	k	t	s	KeyGen	Sign	V	$\lvert\sigma\rvert$	$\lvert$pk$\rvert$
8191	400	80	41	8	9.20	763.02	103.47	4.70	325.00
			32	16	15.87	588.68	39.776	3.90	650.00
			26	32	29.51	507.69	15.98	3.27	1300.00
16381	320	80	41	8	10.16	3.66	2.25	5.04	262.50
			32	16	18.67	2.89	1.76	4.19	525.00
			26	32	35.70	2.35	1.35	3.52	1050.00
8191	248	112	41	8	10.06	2.82	1.42	6.52	193.38
			32	16	19.50	2.23	1.06	5.43	386.75
			26	32	38.48	1.82	0.82	4.55	773.50
16381	274	96	41	8	11.96	2.72	1.99	6.02	233.62
			32	16	23.12	2.11	1.54	5.01	467.25
			26	32	44.35	1.74	1.24	4.20	934.50
Original Protocol ($s = 1$)									
127	272	136	133	1	2.30	0.97	0.60	8.30	27.80
			256	1	2.31	1.83	1.40	5.10	27.80
			512	1	2.32	3.70	2.94	4.40	27.80
			768	1	2.31	5.39	4.50	4.10	27.80
			4096	1	2.32	28.80	24.40	3.60	27.80

Table 2: Performances of our proposed scheme and the original protocol [2]. The first half refers to parameters that are provably secure according to our security analysis, the second half represent a less conservative approach. Times are in MCycles, sizes are in kB.

drawback can also be avoided by increasing q, such that vectors have all distinct entries with higher probability, as it is possible to notice from the second and fourth rows.

Notice that the cases $(n, k) = (248, 112)$ and $(n, k) = (274, 96)$ do not meet our theoretical bounds regarding the uniqueness of π; thus, we can only conjecture its security. Indeed, as previously mentioned, while the uniqueness of π is required for our theoretical analysis to hold, no attacks are currently known. In particular, the number of codewords having the same multi-set is still extremely small, so finding them should be hard; thus, an attack leveraging them is unlikely. Finally, the parameters are such that known attacks on PKP and PEP remain computationally infeasible.

4.3 A less conservative approach

As shown in Table 2, the proposed protocol yields somewhat unappealing performance: while we successfully and securely integrate the multi-key optimization

into the SPECK framework, achieving a faster computational time and very similar signature size, this comes at the expense of significantly larger public keys. The primary cause for this overhead is the requirement for a substantially larger field size q compared to the original design, which increases the representation size of a single parallel repetition and, more importantly, cause a big computational overhead and thus cause a large number of keys (16 or 32) to obtain competitive performances.

Nevertheless, as shown in Table 1, a substantial gap remains between the best key recovery attack and the best forgery attack, with the latter being roughly four times slower. This disparity suggests that our parameter selection may be too conservative. Consequently, we conjecture that permitting spurious solutions could enable a more favorable choice of parameters, thereby improving overall efficiency. Although this relaxation lowers the security margin against adversaries solving PKP, thanks to the big gap highlighted in Table 1, we conjecture that the scheme remains secure.

Specifically, we propose parameter instances that allow a codeword $\mathbf{c}$ to have not all distinct values, but only $d \leqslant n$. This implies that such a random codeword will, on average, have each element repeated n/d times. This implies that there are $n/d!^d$ permutations that do not change the vector, and therefore can be considered a spurious solution. Under this relaxation, Equation (3) is modified to:

$$p_{q,n,k} = \Pr(Z \geqslant 1) \leqslant \left(n! - 1 - \left(\frac{n}{d}! \right)^d \right) q^{k-n} \tag{8}$$

By extending the analysis from Section 3.2 to this setting, we obtain the performance presented in Table 3. Even if also with this set of parameters PEP

q	n	k	d	t	s	$\lvert\sigma\rvert$	$\lvert\mathsf{pk}\rvert$	$\mathrm{CF}_{\mathrm{MITM}}$	$\mathrm{LB}_{\mathrm{CS}}$
1021	320	96	275	41	8	4.49	214.9	143	166
				32	16	3.69	429.9		
				26	32	3.08	859.8		
1021	400	80	331	41	8	3.76	255.9	146	176
				32	16	3.09	525.51		
				26	32	2.58	1023.00		

Table 3: Performance of our proposed scheme with relaxed parameter sets for which the uniqueness of the solution is not guaranteed.

solvers have a time complexity greater that 128 bit, the fact that the uniqueness of the solution is not guaranteed makes evaluating PKP attacks much harder, since this is not the usual setting used by KMP and similar attacks. We leave

a detailed security analysis as a future research direction, as it is highly non trivial.

5 Ring Signature

We now describe how it is possible to obtain a ring signature from the PECK assumption. We first provide the basic definitions for ring signatures; for further details, we refer the reader to [9].

Definition 2 (Ring Signature). *A Ring Signature scheme Π_{RS} is a protocol between r (potential) signers $S_1, \ldots, S_r$ and a verifier. The protocol is composed of the following procedures:*

- $\mathsf{KeyGen}(\mathtt{pp})$*: a probabilistic algorithm that on input the public parameter, output a secret key* sk_i *and a public key* pk_i *for each signer* S_i*. The set* $\mathsf{PK} = \{\mathsf{pk}_i\}_{[s]}$ *of all public key is often called the* ring *of the public key.*
- $\mathsf{Sign}(\mathsf{PK}, \mathsf{sk}_i, \mathtt{Msg})$*: a probabilistic algorithm that on input the set of public keys* PK*, a secret key* sk_i *and a message* Msg*, output a signature* σ*.*
- $\mathsf{V}(\mathsf{PK}, \sigma, \mathtt{Msg})$*: a deterministic algorithm on input the set of public keys* PK*, a message* Msg *and a signature* σ*, it outputs either 1 (accept) if the signature is valid or 0 (reject) if it is not.*

The key feature of a ring signature is that anyone in the ring can produce a valid signature on behalf of the entire ring, and the verifier can only check validity with respect to the whole ring but will not know exactly which secret key was used. We require a ring signature to satisfy the following informal properties:

- *Correctness* honestly generated signatures are always accepted.
- *Anonymity* it is impossible for a verifier to tell which secret key was used to sign.
- *Unforgeability* it is impossible to produce a valid signature without knowing at least one of the secret keys in the ring.

When proving the security of our protocol in Section B we provide detailed definitions about the above property.

5.1 SPECK Ring Signature

We now show how to obtain a ring signature based on SPECK, following a similar structure as [6, 9]. Let $\mathscr{C}_0$ be a public code with generator matrix $\mathbf{G}_0$. To generate its own key, the user i picks randomly a permutation $\mathbf{P}_i$, computes $\mathbf{G}_i = \mathsf{SF}(\mathbf{G}_0\mathbf{P}_i)$. The secret key is $\mathbf{P}_i$ while the public key is $\mathbf{G}_i$. Let $\mathsf{PK} = \{\mathbf{G}_1, ..., \mathbf{G}_s\}$ be the ring of public keys corresponding to the user $U_1, ..., U_s$. Let U_{i*} be the signing user, the ring signature is obtained by parallel repetitions of the following:

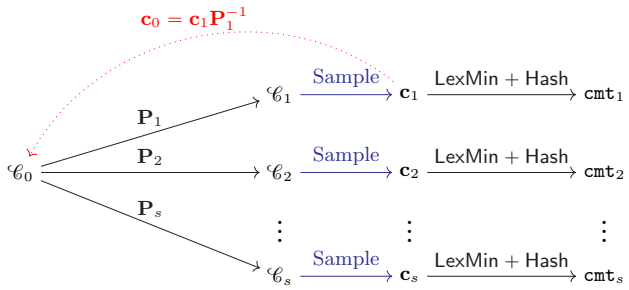

Fig. 5: Representation of a round of a ring signature. In blue the response for $\mathtt{ch} = 0$. The response for $\mathtt{ch} = 1$ is $\mathbf{c}_0$, computed as showed in red.

- from each key in PK, U_{i*} samples a codeword $\mathbf{c}_i \xleftarrow{\$} \mathscr{C}_i$. Let $\{\mathtt{Seed}_i\}_{[s]}$ be the seeds used for the sampling.
- For all $\mathbf{c}_i$ with $i = 1, ..., s$, U_{i*} computes $\mathtt{cmt}_i = \mathsf{Hash}(\mathsf{LexMin}(\mathbf{c}_i))$.
- Let $b \in \{0, 1\}$ be the challenge bit:
  - if $b = 0$, then the response is $\mathsf{rsp} = \{\mathtt{Seed}_i\}_{[s]}$;
  - if $b = 1$, then the response is $\mathsf{rsp} = \mathbf{c}_0 = \mathbf{c}_{i*}\mathbf{P}_{i*}^{-1}$.
- the final signature is $(\mathsf{rsp}, \{\mathtt{cmt}_i\}_{[s]})$. Notice that before sending all the $\mathtt{cmt}_i$, U_{i*} shuffles them to ensure anonymity.

To verify the signature, the verifier does the following

- if $b = 0$, the verifier parses the response as $\mathsf{rsp} = \{\mathtt{Seed}_i\}_{[s]}$, then samples $\mathbf{c}_i \xleftarrow{\mathtt{Seed}_i} \mathscr{C}_i$. Then checks that $\mathtt{cmt}_i = \mathsf{Hash}(\mathsf{LexMin}(\mathbf{c}_i))$ for $i = 1, ..., s$.
- if $b = 1$, the verifier parses the response as $\mathsf{rsp} = \mathbf{c}_0$ then checks that $\mathbf{c}_0 \in \mathscr{C}_0$ and that there exists a i^* such that $\mathtt{cmt}_{i*} = \mathsf{Hash}(\mathsf{LexMin}(\mathbf{c}_0))$.

Notice that the attacks on SPECK with multiple keys do not impact the security of the ring signature. Indeed, the challenge space is still binary and finding $\tilde{\mathbf{c}}_i \in \mathscr{C}_i, \tilde{\pi}_i \in S_n$ such that $\tilde{\pi}(\tilde{\mathbf{c}}) \in \mathscr{C}_j$ for some i, j does not provide any advantage to an eventual attacker. For this reason we can use the same parameter sets as the original SPECK.

To achieve more compact signatures we can use the same optimization as [9] and [6]. Namely, fixed-weight challenges and compressing all the commitments $\mathtt{cmt}_i$ into an index-hiding Merkle-tree so that it is possible to include the root into the signature plus a path to $\mathtt{cmt}_{i*}$ when $b = 1$. Moreover we can also compress all the seeds into a single seed tree, avoiding sending s seeds for $b = 0$.

Figure 5 shows the schematic idea of a single round, while Figure 6 shows the detailed description of the scheme, where HMerkleTree, GetPathHMT and HMerkleTreeFromPath are the algorithm used to, respectively, generate an index-hiding Merkle-Tree from the leaves, compute the path starting from a leaf and compute the Merkle-tree starting from a path and a leaf. For more information about index-hiding Merkle-tree see [9].



Fig. 6: SPECK ring signature scheme

Security Proof The security proof of our protocol is quite straightforward and very similar to the one of [6, 9]. We included it in Section B.

5.2 Performance analysis

We now provide a brief outline about the performance of our protocol, and compare it [6]. As showed in [6], the signature size is computed as:

$$\underbrace{w\lceil \log t \rceil}_{\mathsf{ch}} + \underbrace{w(n\lceil \log n \rceil + n\lceil \log q \rceil + 2\lambda\lceil \log s \rceil + \lambda)}_{\{\mathsf{rsp}_i\}} + \underbrace{\lambda\lceil \log t \rceil}_{\{\mathsf{Seed}\}} \tag{1}$$

where s denotes the number of keys in the ring.

Table 4 shows the signature size for various ring sizes. As expected, our signature is slightly bigger compared to the one in [6], however it is expected to be much faster. Indeed, each round of [6] requires s Gaussian eliminations to produce all the commitments, while our proposal avoids them. As in the case of SPECK and LESS, we expect our protocol to be up to two orders of magnitude faster than [6]. [4]

Finally, it is important to notice that [6] predates the canonical form optimization of LESS, thus it is likely that an improved version with shorter signatures could be possible.

Instance	q	n	k	t	w	s	$\lvert\mathsf{pk}\rvert$	$\lvert\sigma\rvert$
This work	127	252	126	512	23	2^3	27.8	13.61
						2^6		15.82
						2^{12}		20.24
LESS [6]	127	230	115	233	31	2^3	11.57	10.76
						2^6		13.74
						2^{12}		19.69

Table 4: Parameters and signature size for various ring dimension. All sizes are in kB.

6 Conclusions

In this paper we improved the analysis of the PECK introduced in [2]. In particular we advanced the cryptanalysis of PECK, finding conditions under which PECK is hard even when one is allowed to choose the codewords, instead of sampling them randomly. This allowed us to solve the issues related to the multi-key optimization that were present in the original paper. Our analysis shows that the multi-key optimization secure instances have performances that are worse compared to the original protocol. For this reason, we propose an alternative set of parameters, that lies outside our bounds but for which still the solution should be unique with very high probability, and thus a security analysis using traditional PKP attacks can be done. While the performances are better, the proposed solution still suffers from incredibly big public keys, making it quite unappealing. For this reason we propose a second relaxation, where we allow

[4] As noted in [3] SPECK public key can be halved at the price of a small computational slowdown. In Table 4 we do not include this optimization.

for multiple solutions. This improves the performances even more, however the security analysis is now way more difficult and we only conjecture the security of the protocol, while we leave a formal analysis for future works.

As a second contribution we obtain a simple ring signature based on SPECK. Compared to [6] that is based on LESS, our ring signature should be much faster, as it is already the case for SPECK when compared to LESS. Unfortunately, when compared to other similar ring signatures, such as the one in [9] our signature lacks the linkability property, which remains an open problem in code-based cryptography, as the solutions proposed in [6] are susceptible to the attacks presented in [11].

Acknowledgements

This work was partially supported by the Italian Ministry of University and Research (MUR) under the Italian Fund for Applied Science (FISA 2022), Call for tender No. 1405 published on 13-09-2022 - project title "Quantum-safe cryptographic tools for the protection of national data and information technology assets" (QSAFEIT) - No. FISA 2022-00618 (CUP I33C24000520001), Grant Assignment Decree no. 15461 adopted on 02.08.2024.

Michele Battagliola is also supported by the University of St. Gallen, Great Mind Fellowship, Project 103 I 638 for "Code-Based Cryptography for Post-Quantum Security (CoB-PQS)".

References

[1] G. Alagic, M. Bros, P. Ciadoux, D. Cooper, Q. Dang, T. Dang, J. Kelsey, J. Lichtinger, Y.-K. Liu, C. Miller, et al. "Status Report on the First Round of the Additional Digital Signature Schemes for the NIST Post-Quantum Cryptography Standardization Process". In: *NIST IR* 8528 (2024).

[2] M. Baldi, M. Battagliola, R. El Mechri, P. Santini, R. Schiavoni, and D. De Zuane. *SPECK: Signatures from Permutation Equivalence of Codes and Kernels*. Cryptology ePrint Archive, Paper 2025/923. 2025. URL: `https://eprint.iacr.org/2025/923`.

[3] M. Baldi, R. E. Mechri, P. Santini, and R. Schiavoni. *On the representation of self-orthogonal codes and applications to cryptography*. Cryptology ePrint Archive, Paper 2025/2279. 2025. URL: `https://eprint.iacr.org/2025/2279`.

[4] M. Bardet, A. Otmani, and M. Saeed-Taha. "Permutation code equivalence is not harder than graph isomorphism when hulls are trivial". In: *2019 IEEE International Symposium on Information Theory (ISIT)*. IEEE. 2019, pp. 2464–2468.

[5] A. Barenghi, J.-F. Biasse, E. Persichetti, and P. Santini. "On the computational hardness of the code equivalence problem in cryptography". In: *Advances in Mathematics of Communications* 17.1 (2023), pp. 23–55.

[6] A. Barenghi, J.-F. Biasse, T. Ngo, E. Persichetti, and P. Santini. "Advanced signature functionalities from the code equivalence problem". In: *International Journal of Computer Mathematics: Computer Systems Theory* 7.2 (2022), pp. 112–128. DOI: 10.1080/23799927.2022.2048206. eprint: https://doi.org/10.1080/23799927.2022.2048206. URL: https://doi.org/10.1080/23799927.2022.2048206.

[7] M. Battagliola, R. Mora, and P. Santini. *Using the Schur Product to Solve the Code Equivalence Problem*. Cryptology ePrint Archive, Paper 2025/1017. 2025. URL: https://eprint.iacr.org/2025/1017.

[8] W. Beullens. "Not enough LESS: an improved algorithm for solving code equivalence problems over F q". In: *International Conference on Selected Areas in Cryptography*. Springer. 2020, pp. 387–403.

[9] W. Beullens, S. Katsumata, and F. Pintore. "Calamari and Falafl: Logarithmic (Linkable) Ring Signatures from Isogenies and Lattices". In: *ASIACRYPT 2020, Part II*. Ed. by S. Moriai and H. Wang. Vol. 12492. LNCS. Springer, Cham, Dec. 2020, pp. 464–492. DOI: 10.1007/978-3-030-64834-3_16.

[10] D. Boneh and V. Shoup. *A Graduate Course in Applied Cryptography*. Version 0.6. Online Manuscript, 2023. URL: https://toc.cryptobook.us/.

[11] A. Budroni, J.-J. Chi-Domínguez, G. D'Alconzo, A. J. D. Scala, and M. Kulkarni. "Don't Use it Twice! Solving Relaxed Linear Equivalence Problems". In: *ASIACRYPT 2024, Part VIII*. Ed. by K.-M. Chung and Y. Sasaki. Vol. 15491. LNCS. Springer, Singapore, Dec. 2024, pp. 35–65. DOI: 10.1007/978-981-96-0944-4_2.

[12] A. Budroni, M. Defranceschi, and F. Pintore. *Recursion Enabled: Improved Cryptanalysis of the Permuted Kernel Problem*. Cryptology ePrint Archive, Paper 2025/2073. 2025. URL: https://eprint.iacr.org/2025/2073.

[13] T. Chou, E. Persichetti, and P. Santini. "On linear equivalence, canonical forms, and digital signatures". In: *Designs, Codes and Cryptography* (2025), pp. 1–43.

[14] L. De Feo and S. D. Galbraith. "SeaSign: Compact Isogeny Signatures from Class Group Actions". In: *EUROCRYPT 2019, Part III*. Ed. by Y. Ishai and V. Rijmen. Vol. 11478. LNCS. Springer, Cham, May 2019, pp. 759–789. DOI: 10.1007/978-3-030-17659-4_26.

[15] W. C. Huffman and V. Pless. *Fundamentals of Error-Correcting Codes*. Cambridge University Press, 2003.

[16] E. Koussa, G. Macario-Rat, and J. Patarin. "On the complexity of the Permuted Kernel Problem". In: *Cryptology ePrint Archive* (2019).

[17] J. Leon. "Computing automorphism groups of error-correcting codes". In: *IEEE Transactions on Information Theory* 28.3 (1982), pp. 496–511.

[18] M. A. Saeed. "Algebraic approach for code equivalence". PhD thesis. Normandie Université; University of Khartoum, 2017.

[19] P. Santini, M. Baldi, and F. Chiaraluce. "Computational Hardness of the Permuted Kernel and Subcode Equivalence Problems". In: *IEEE Trans.*

Inf. Theor. 70.3 (Mar. 2024), 2254–2270. DOI: 10.1109/TIT.2023.3323068. URL: https://doi.org/10.1109/TIT.2023.3323068.

[20] N. Sendrier. "Finding the permutation between equivalent linear codes: The support splitting algorithm". In: *IEEE Transactions on Information Theory* 46.4 (2000), pp. 1193–1203.

A Signature Protocol

Figure 7 shows our signature scheme, which is obtained by applying the Fiat-Shamir transform to the previously explained interactive protocol. As in the original work of SPECK [2] we included the lexicographical order optimization to avoid the ephemeral permutation $\mathbf{P}^*$. This allows us to simply send codewords instead of permutations. Moreover, each codeword is compressed sending only the first k entry, while the other $n-k$ are computed from them by the verifier.

B Security Proof of the Ring Signature

We first recall the formal security definitions for ring signatures.

More formally we have the following definitions:

Definition 3 (Anonymity). *A ring signature scheme Π_{RS} is anonymous (against full key exposure) if, for all $\lambda \in \mathbb{N}$ and $s = \mathsf{poly}(\lambda)$, any PPT adversary $\mathcal{A}$ has at most negligible advantage in the following game played against a challenger.*

1. *The challenger generates key pairs $(\mathsf{pk}_i, \mathsf{sk}_i) = \mathsf{KeyGen}(\mathsf{pp})$ for all $i \in [s]$. It also samples a random bit $b \leftarrow \{0,1\}$;*
2. *The challenger provides pp and r to $\mathcal{A}$;*
3. *$\mathcal{A}$ outputs a challenge $(\mathsf{PK}, \mathtt{Msg}, i_0, i_1)$ to the challenger, where the ring PK must contain pk_{i_0} and pk_{i_1}. The challenger then runs $\sigma^* \leftarrow \mathsf{Sign}(\mathsf{PK}, \mathsf{sk}_{i_b}, \mathtt{Msg})$, and provides σ^* to $\mathcal{A}$;*
4. *$\mathcal{A}$ outputs a guess $\mathtt{ch}^*$ and wins if $\mathtt{ch}^* = b$.*

The advantage of $\mathcal{A}$ is defined as

$$\mathcal{A}_{\mathsf{RS}}^{\mathsf{Anon}}(\mathcal{A}) = |\Pr[\mathcal{A} \text{ wins}] - 1/2|.$$

Definition 4 (Unforgeability). *A ring signature scheme Π_{RS} is unforgeable (with respect to insider corruption) if, for all $\lambda \in \mathbb{N}$ and $s = \mathsf{poly}(\lambda)$, any PPT adversary $\mathcal{A}$ has at most negligible advantage in the following game played against a challenger.*

1. *The challenger generates key pairs $(\mathsf{PK}, \mathsf{sk}_i) = \mathsf{KeyGen}(\mathsf{pp})$ generating $(\mathsf{sk}_i, \mathsf{pk}_i)$ for all $i \in [s]$. It sets $\mathsf{PK} := \{\mathsf{pk}_i\}_{i \in [s]}$ and initializes two empty sets SL and CL.*
2. *The challenger provides pp and PK to $\mathcal{A}$;*



Fig. 7: The SPECK signature scheme using the multi-key optimization

3. *$\mathcal{A}$ can make signing and corruption queries an arbitrary polynomial number of times:*
   - $(\mathsf{sign}, i, \mathtt{Msg}, \mathsf{PK})$*: The challenger checks if* $\mathsf{pk}_i \in R$ *and if so it computes the signature* $\sigma \leftarrow \mathsf{Sign}(\mathsf{sk}_i, \mathtt{Msg}, \mathsf{PK})$*. The challenger provides* σ *to* $\mathcal{A}$ *and adds* $(i, \mathtt{Msg}, \mathsf{PK})$ *to* SL*;*
   - $(\mathsf{corrupt}, i)$*: The challenger adds* pk_i *to* CL *and returns* sk_i *to* $\mathcal{A}$*.*
4. *$\mathcal{A}$ outputs* $(\mathsf{PK}^*, \mathtt{Msg}^*, \sigma^*)$*. We say the adversary* $\mathcal{A}$ *wins if* $\mathsf{PK}^* \subset \mathsf{PK} \backslash \mathsf{CL}$*,* $(\cdot, \mathtt{Msg}^*, \mathsf{PK}^*) \notin \mathsf{SL}$*, and* $\mathsf{V}(\mathsf{PK}^*, \mathtt{Msg}^*, \sigma^*) = 1$*.*

The advantage of $\mathcal{A}$ *is defined as*

$$\mathcal{A}_{\mathsf{RS}}^{\mathsf{Unf}}(\mathcal{A}) = \Pr[\mathcal{A} \textit{ wins}].$$

We now prove the security of our ring signature. The proofs are similar to the one presented in [9, Appendix A]. We start with a preliminary lemma that tries to capture the idea of Honest-Verifier Zero-Knowledge in the context of ring signature. In particular, given $\mathbf{G}_0$ a generator matrix and $\mathsf{PK} = \{\mathbf{G}_1, ..., \mathbf{G}_s\}$ a ring of public keys we show that there exists a simulator $\mathcal{S}$ that on input an index i^* and PK outputs a valid signature with the same distribution as an honestly generated one.

Lemma 1. *Let $\mathbf{G}_0$ be a generator matrix and $\mathsf{PK} = \{\mathbf{G}_1, ..., \mathbf{G}_s\}$ a ring of public keys. Then there exits a simulator $\mathcal{S}$ such that for all polynomial time algorithm $\mathcal{D}$*

$$\Pr[\mathcal{D}(\sigma \leftarrow \mathcal{S}(\mathbf{G}_0, \mathsf{PK}, i^*)) = 1] - \Pr[\mathcal{D}(\sigma \leftarrow \mathsf{Sign}(\mathbf{G}_0, \mathsf{PK}, \mathbf{P}_{i*})) = 1] = \eta(\lambda)$$

where $\eta(\lambda)$ is negligible in λ.

Proof. The proof is basically the same as the Honest-Verifier Zero-Knowledge of the base protocol. In particular $\mathcal{S}$ picks a random challenge $\mathtt{ch}$. Then for all the bit $\mathtt{ch}_i$, it does the following:

- if $\mathtt{ch}_i = 0$ then $\mathcal{S}$ follows the protocol normally,
- if $\mathtt{ch}_i = j^*$ with $j^* \neq 0$ then $\mathcal{S}$ picks a random codeword $\mathbf{c} \in \mathscr{C}_0$, compute $\mathtt{cmt}^{(i,j^*)} = \mathsf{Hash}(\mathsf{LexMin}(\mathbf{c})||\mathsf{Salt}||i||\mathtt{Msg})$ and then sets all the other $\mathtt{cmt}^{(i,j)}$ as random string. Then it follows the protocol normally.

Finally, it sets the challenge as $\mathtt{ch}$. The resulting signature verifies by construction and it is indistinguishable from an honestly generated signature.

As a direct consequence of Lemma 1 we have that the signature satisfies the anonymity of property Definition 3

Proposition 2. *If the* PECK *problem is hard then the ring signature of Figure 6 is anonymous.*

Proof (Sketch). The proof is a simple corollary of Lemma 1. In particular the challenger can run $\mathcal{S}$ instead of the signing algorithm, thus avoiding directly the usage of the secret key. It trivially follows that the probability of winning such a game is $\frac{1}{2}$.

Finally, we show the unforgeability property of Definition 4.

Proposition 3. *If the* PECK *problem is hard and q lies in the* High *q regime, then the ring signature of Figure 6 is unforgeable.*

Proof. The proof is a consequence of the special-soundness property of SPECK when instantiated in the high q regime (see [2]). In particular the challenger receives a PEP challenge $\mathbf{G}_0, \mathbf{G}_1$, then creates a ring PK of s keys creating $s-1$ other codes equivalent to $\mathbf{G}_0$, WLOG $\mathsf{PK} = \{\mathbf{G}_1, ..., \mathbf{G}_s\}$. The challenger then

sends PK to the forger $\mathcal{A}$ of the unforgeability game of Definition 4. To simulate the signing query the challenger uses the same strategy of Lemma 1, while it can respond to all the corruption queries except for $i = 1$. In this case the challenger aborts.

If the challenger does not abort, let $(\mathsf{PK}^*, \mathtt{Msg}^*, \mathsf{rsp}^*)$ be the forgery produced, with $\mathsf{rsp} = (\mathtt{Salt}, \mathtt{cmt}, \mathtt{Seeds}, \{\mathbf{y}_1^{(i)}, \mathtt{path}^{(i)}\}_{i \in J})$. Then the challenger rewinds the forger and changes the challenges corresponding to $\mathsf{Hash}(\mathtt{cmt}||\mathtt{Salt}||\mathtt{Msg})$. Thanks to the forking lemma the forger will produce a second forgery with non-negligible probability.

The challenger is able to extract the solution for the PEP instance as long as:

- the forger does not ask for the corruption of the user 1. Since all the keys are identically distributed this happens with probability at least $\frac{1}{s}$, depending on the number of corruption queries.
- the forger produces a forgery corresponding to the key of the first player in both the executions. This happens with probability at least $\frac{1}{s^2}$.

Overall the security loss is bounded by $\frac{1}{s^3}$ which is non negligible. So we must admit that the advantage of the forger is negligible.

C Sampling

We provide here a brief description of our sampling algorithm. Let $\mathbf{G}$ be the generator matrix of a $[n, k]-$code $\mathscr{C}$, as explained in Section 3 we need to sample $\mathbf{c} \in \mathscr{C}$ with all different entries. To do so, we first sample a random $\mathbf{u} \in \mathbb{F}_q^k$ with all different entries, then we compute $\mathbf{uG}$ and check whether or not it has repeated entries. If it is the case we add linear combinations of the rows of $\mathbf{G}$ until the vector is free from repetition. It is immediate to see that by growing q the probability of $\mathbf{uG}$ having all different entries increases. It is more subtle to understand the impact of k: indeed, by construction the first k entries are all different if $\mathbf{G}$ is in systematic form. Thus, only $n - k$ need to be checked. It turns out that increasing k drastically decreases the probability of $\mathbf{uG}$ having repeated entries, for this reason instances with higher k are more efficient. The expected number of resamplings, given in Section C, needed can be estimated as the inverse of the following probability:

$$\Pr_{\mathsf{Sampling}} = \prod_{i=0}^{n-k-1} \frac{q-k-i}{q} = \prod_{i=0}^{n-k} 1 - \frac{k-i}{q} = \frac{q!}{q-k-i!} \cdot q^{k-n} \tag{9}$$

On the other hand Section C shows the iterations needed for the set of parameters proposed in *Table* 3, d was chosen to minimize the number of samples required to select a codeword with that specific number of entries. In particular,

q	n	R	k	$\Pr_{\mathsf{Sampling}}$
8191	400	0.2	80	2^{-14}
8191	320	0.25	80	2^{-9}
8191	274	0.35	96	2^{-6}
8191	248	0.45	112	2^{-4}
16381	400	0.2	80	2^{-7}
16381	320	0.25	80	2^{-4}
16381	274	0.35	96	2^{-3}

Table 5: Probability of sampling a codeword with no repeated entries for the proposed parameter sets

the average cardinality of $\mathscr{C}^{\mathsf{d}}_{\mathsf{diff}}$, the set of codewords within a random code with d distinct values, can be estimated using the principle of inclusion-exclusion as:

$$|\mathscr{C}^{\mathsf{d}}_{\mathsf{diff}}| = \binom{q}{d} \sum_{j=0}^{d} (-1)^j \binom{d}{j} (-d-j)^n$$

Then the probability of sampling a random codeword with d distinct values can be estimated as:

$$\Pr_{\mathsf{Sampling,d}} = \frac{|\mathscr{C}^{\mathsf{d}}_{\mathsf{diff}}|}{q^n} q^k = \frac{\binom{q}{d} \sum_{j=0}^{d} (-1)^j \binom{d}{j} (-d-j)^n}{q^n} q^k$$

q	n	R	k	d	$\Pr^{\mathsf{d}}_{\mathsf{Sampling}}$	
1021	320	0.3	96	275		2^{-4}
1021	400	0.2	80	331		$2^{-3.5}$

Table 6: Probability of sampling a codeword with d distinct entries entries for the proposed parameter sets