PERK

Principal submitters:

Najwa Aaraj, Technology Innovation Institute, UAE
Slim Bettaieb, Technology Innovation Institute, UAE
Loïc Bidoux, Technology Innovation Institute, UAE
Alessandro Budroni, Technology Innovation Institute, UAE
Victor Dyseryn, XLIM, University of Limoges, France
Andre Esser, Technology Innovation Institute, UAE
Philippe Gaborit, XLIM, University of Limoges, France
Mukul Kulkarni, Technology Innovation Institute, UAE
Victor Mateu, Technology Innovation Institute, UAE
Marco Palumbi, Technology Innovation Institute, UAE
Lucas Perin, Technology Innovation Institute, UAE
Jean-Pierre Tillich, INRIA, Paris, France

Inventors/Developers: Same as submitters

Owner: Same as submitters

Main contact: Loïc Bidoux
loic.bidoux@tii.ae
+971-552-420-042
Technology Innovation Institute
P.O.Box: 9639, Masdar City
Abu Dhabi, United Arab Emirate

Backup contact: Philippe Gaborit
philippe.gaborit@unilim.fr
+33-5-87-50-67-81
University of Limoges
123 avenue Albert Thomas
87 060 Limoges Cedex, France

Version: 1.0

Release date: 31/05/2023

Table of Contents

1 Introduction

PERK is a digital signature scheme that is designed to provide security against attacks which may use quantum computers, as well as attacks by classical computers. The scheme builds on a zero-knowledge proof of knowledge system based on the conjectured post-quantum security of a variant of the Permuted Kernel Problem (PKP), and hash functions modelled as random oracles. The zero-knowledge proof is constructed from the well established MPC-in-the-head paradigm, and it is then converted into a signature scheme using the Fiat-Shamir transform in the random oracle model. The name of the scheme stems from the difficult problem based on PERmuted Kernels, at the core of the security of the protocol.

We present notations and conventions used in this document in Section 2.1, followed by the definitions of standard cryptographic primitives in Section 2.2. We provide details about proof of knowledge systems and their properties in Section 2.3. We then present definition of signature schemes in Section 2.4, followed by the background on random oracle model and Fiat-Shamir transform in Section 2.4.2. We conclude Section 2 by presenting the new definitions of the variants of the Permuted Kernel Problem (PKP), which are introduced by us in Section 2.5. We provide the constructions of our zero-knowledge proof of knowledge system in Section 3.1 and follow it up with construction of PERK signature scheme in Section 3.2. In Section 4 we give details regarding the parameter sets for different desired security levels. We provide the implementation details along with various benchmarks and performance results in Section 5. We present the formal security proofs for the zero-knowledge proof of knowledge system and PERK signature scheme in Section 6.1 and Section 6.2 respectively. We discuss the security of our proposal in the context of known attacks in Section 7.1 and Section 7.2, and analyze the concrete security estimates of our underlying hardness assumption (variant of PKP) in Section 7.3. We conclude by discussing the advantages and limitations of our proposal in Section 8.

2 Preliminaries

2.1 Notations and Conventions

For integers a, b we denote $[a, b]$ the set of integers i such that $a \leq i \leq b$. We write $[n]$ as a shorthand for $[1, n]$. We denote $\mathcal{S}_n$ the group of permutations of the set $[n]$. Let $\mathbb{F}_q$ denote the finite field of q elements where q is the power of a prime. Vectors are denoted by bold lower-case letters and matrices by bold capital letters (e.g., $\mathbf{v} = (v_1, \ldots, v_n) \in \mathbb{F}_q^n$ and $\mathbf{M} = (m_{ij})_{\substack{1 \leqslant i \leqslant k \\ 1 \leqslant j \leqslant n}} \in \mathbb{F}_q^{k \times n}$).

If S is a finite set, we denote by $x \xleftarrow{\$} S$ that x is chosen uniformly at random from S. Similarly, we write $x \xleftarrow{\$,\theta} S$, if x is sampled pseudo-randomly from the set S, based on the seed θ.

We use x to denote input and denote its length by $|x|$. We use λ to denote the security parameter. We call a function $f : \mathbb{N} \to \mathbb{R}^+$ *negligible*, if for all $c \in \mathbb{N}$ there exists a $N_0 \in \mathbb{N}$ such that $f(n) < 1/n^c$ for all $n > N_0$. We write $\mathsf{negl}(\lambda)$ to denote an arbitrary negligible function. We use $\mathsf{poly}(\lambda)$ for function which is *polynomially bounded* in λ, that is there exists $c, \lambda_0 \in \mathbb{N}$ such that $\mathsf{poly}(\lambda) \leq \lambda^c$ for all $\lambda \geq \lambda_0$. We also abbreviate *probabilistic polynomial-time* as PPT.

Let X and Y be two discrete random variables defined over a finite support D. The *statistical distance* between the two distributions is defined as

$$\Delta(X, Y) := \frac{1}{2} \sum_{d \in D} |\Pr[X = d] - \Pr[Y = d]|.$$

We say two ensembles of random variables $\{X_\lambda\}_{\lambda \in \mathbb{N}}$, $\{Y_\lambda\}_{\lambda \in \mathbb{N}}$ are *statistically close* if there exists a negligible function $\mathsf{negl} : \mathbb{N} \to \mathbb{R}^+$ such that $\Delta(X_\lambda, Y_\lambda) \leq \mathsf{negl}(\lambda)$ for all $\lambda \in \mathbb{N}$. We say two ensembles of random variables $\{X_x\}_{x \in \{0,1\}^*}$, $\{Y_x\}_{x \in \{0,1\}^*}$ are *statistically close* if there exists a negligible function $\mathsf{negl} : \mathbb{N} \to \mathbb{R}^+$ such that $\Delta(X_x, Y_x) \leq \mathsf{negl}(|x|)$ for all $x \in \{0,1\}^*$.

2.2 Standard Cryptographic Primitives

2.2.1 Pseudorandom Generators

Definition 2.1 (Pseudorandom Generator (PRG)). *Let p be a polynomial and let G be a deterministic polynomial-time algorithm such that for any $\lambda \in \mathbb{N}$ and any input $s \in \{0,1\}^\lambda$, the result $\mathsf{G}(s)$ is a string of length $\mathsf{p}(\lambda)$. We say that G is a* pseudorandom generator *if the following conditions hold:*

1. *Expansion: For every $\lambda \in \mathbb{N}$ it holds that $\mathsf{p}(\lambda) > \lambda$.*
2. *Pseudorandomness: For any PPT algorithm D, there is a negligible function negl such that*

$$|\Pr[\mathsf{D}(\mathsf{G}(s)) = 1] - \Pr[\mathsf{D}(r) = 1]| \leq \mathsf{negl}(\lambda)$$

where the first probability is taken over the uniform choice of $s \in \{0,1\}^\lambda$ and the randomness of D, and the second probability is taken over the choice of $r \in \{0,1\}^{\mathsf{p}(\lambda)}$ and the randomness of D.

We say G is $(\mathsf{t}, \epsilon_{\mathsf{PRG}})$-secure if for every D running in time at most $\mathsf{t}(\lambda)$ the success probability of D is upper bounded by some function $\epsilon_{\mathsf{PRG}}(\lambda)$.

2.2.2 Collision-Resistant Hash Functions

Definition 2.2 (Collision-Resistant Hash Functions (CRHF)). *Let ℓ, κ be polynomials and and let $\mathcal{H} = \{\mathsf{H}_k : \{0,1\}^* \to \{0,1\}^{\ell(\lambda)};\ k \in \{0,1\}^{\kappa(\lambda)}\}_\lambda$ be a family of functions indexed by $\lambda \in \mathbb{N}$. We say that $\mathcal{H}$ is* collision-resistant *if there exists a negligible function negl such that, for any PPT algorithm $\mathcal{A}$ it holds that,*

$$\Pr\left[\begin{array}{l} x \neq x' \bigwedge \\ \mathsf{H}_k(x) = \mathsf{H}_k(x') \end{array} \;\middle|\; \begin{array}{l} k \xleftarrow{\$} \{0,1\}^{\kappa(\lambda)}; \\ (x, x') \longleftarrow \mathcal{A}(k) \end{array}\right] \leq \mathsf{negl}(\lambda).$$

2.2.3 Commitments Schemes

Definition 2.3 (Commitment Scheme). *A commitment scheme is a tuple of algorithms* $(\mathsf{Com}, \mathsf{Open})$ *such that* $\mathsf{Com}(r, m)$ *returns a commitment* c *for the message* m *and randomness* r *while* $\mathsf{Open}(c, r, m)$ *returns either* 1 *(*accept*) or* 0 *(*reject*). A commitment scheme is said to be correct if:*

$$\mathsf{Pr}\left[b = 1 \;\middle|\; c \leftarrow \mathsf{Com}(r, m),\ b \leftarrow \mathsf{Open}(c, r, m)\right] = 1.$$

Definition 2.4 (Computationally Hiding). *Let* (m_0, m_1) *be a pair of messages, the advantage of* $\mathcal{A}$ *against the hiding experiment is defined as:*

$$\mathsf{Adv}^{\mathsf{hiding}}_{\mathcal{A}}(1^\lambda) = \left| \mathsf{Pr}\left[b = b' \;\middle|\; \begin{array}{l} b \xleftarrow{\$} \{0,1\},\ r \xleftarrow{\$} \{0,1\}^\lambda \\ c \longleftarrow \mathsf{Com}(r, m_b),\ b' \longleftarrow \mathcal{A}.\mathsf{guess}(c) \end{array} \right] - \frac{1}{2} \right|.$$

A commitment scheme is computationally hiding if for all PPT *adversaries* $\mathcal{A}$ *and every pair of messages* (m_0, m_1), $\mathsf{Adv}^{\mathsf{hiding}}_{\mathcal{A}}(1^\lambda)$ *is negligible in* λ.

We say Com is $(\mathsf{t}, \epsilon_{\mathsf{Com}})$-secure if for every $\mathcal{A}$ running in time at most $\mathsf{t}(\lambda)$ the success probability of $\mathcal{A}$ is upper bounded by some function $\epsilon_{\mathsf{Com}}(\lambda)$.

Definition 2.5 (Computationally Binding). *The advantage of an adversary* $\mathcal{A}$ *against the commitment binding experiment is defined as:*

$$\mathsf{Adv}^{\mathsf{binding}}_{\mathcal{A}}(1^\lambda) = \mathsf{Pr}\left[\begin{array}{l} m_0 \neq m_1 \\ 1 \longleftarrow \mathsf{Open}(c, r_0, m_0) \\ 1 \longleftarrow \mathsf{Open}(c, r_1, m_1) \end{array} \;\middle|\; (c, r_0, r_1, m_0, m_1) \longleftarrow \mathcal{A}.\mathsf{choose}(1^\lambda) \right].$$

A commitment scheme is computationally binding if for all PPT *adversaries* $\mathcal{A}$, $\mathsf{Adv}^{\mathsf{binding}}_{\mathcal{A}}(1^\lambda)$ *is negligible in* λ.

2.2.4 Merkle Trees

Merkle trees can be used in our context to compress randomness seeds as suggested in [KKW18]. Suppose a party needs to generate N seeds and then to send only $N-1$ of those seeds (without knowing in advance which seed should not be sent). The principle is to build a binary tree of depth $\lceil \log_2(N) \rceil$. The root of the tree is labeled with a master seed θ. The rest of the tree is labeled inductively by using a PRG of double extension on each parent node and splitting the output on the left and right children.

To reveal all seeds except seed number $i \in [N]$, the principle is to reveal the labels on the siblings of the paths from the root of the tree to leave i. It allows to reconstruct all seeds but seed number i at the cost of communicating $\lfloor \log_2(N) \rfloor$ labels, which is more effective than communicating $N-1$ seeds.

2.3 Proofs of Knowledge

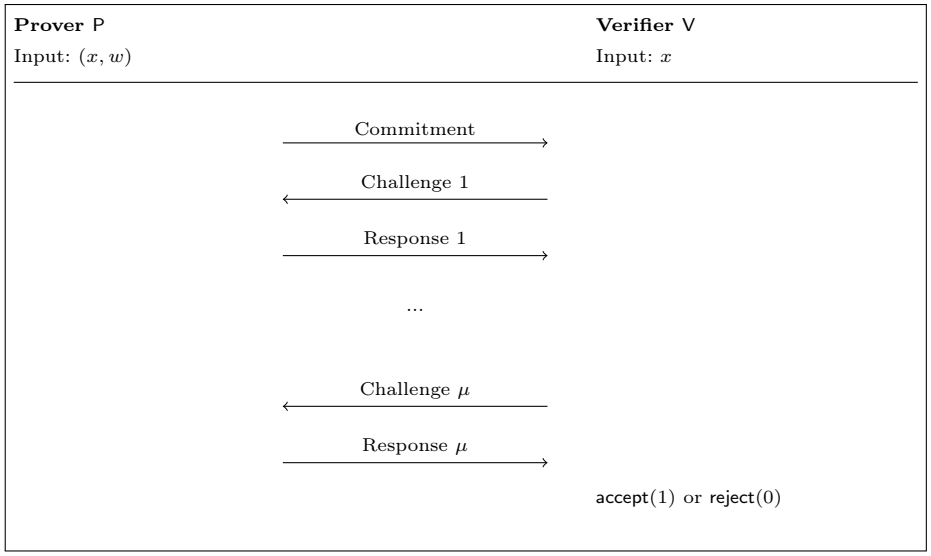
2.3.1 Zero-Knowledge Proofs of Knowledge

Let $R \subseteq \{0,1\}^* \times \{0,1\}^*$ be a binary relation. If $(x;w) \in R$, we say x is a *statement* and w is a *witness* for x. The set of valid witnesses for x is denoted by $R(x) = \{w \,|\, (x;w) \in R\}$. A statement that admits a witness is called a *true* or *valid* statement. The set of true statements is denoted by $L_R := \{x : \exists\, w \text{ such that } (x;w) \in R\}$. A binary relation is said to be an NP relation if the validity of a witness w can be verified in time polynomial in the size $|x|$ of the statement x. From now on we assume all relations to be NP relations.

An interactive proof for relation R aims for a prover P to convince a verifier V that a statement x admits a witness, or even that the prover *knows* a witness $w \in R(x)$.

Definition 2.6 (Interactive proof (cf. [AF22])). *An interactive proof $\Pi = (\mathsf{P}, \mathsf{V})$ for relation R is an interactive protocol between two probabilistic machines, a prover P and a polynomial time verifier V. Both P and V take as public input a statement x and, additionally, P takes as private input a witness $w \in R(x)$, which is denoted as $(\mathsf{P}(w), \mathsf{V})(x)$. The verifier V either accepts or rejects the prover's claim of knowing a witness for x, this decision by the verifier is considered the output of the protocol. The set of all messages exchanged in the protocol execution is called a transcript and is denoted $\langle \mathsf{P}(x,w), \mathsf{V}(x) \rangle$. We call the either* accepting *(or resp.* rejecting*) based on whether the verifier accepts (or rejects) the prover's claim.*

We assume that the prover sends the first and the last message in any interactive proof. Hence, the number of messages is always an odd number $2\mu + 1$. We also say Π is a $(2\mu+1)$-round proof. It is represented in the following figure.



An interactive proof Π is *complete* if the verifier V accepts honest executions with a public-private input pair $(x; w) \in R$ with high probability. It is *sound* if the verifier rejects the false statements $x \notin L_R$ with high probability. In this work, we follow the presentation of [AF22] and do not require these properties as part of definition of interactive proofs, but consider them as desirable additional security properties.

Definition 2.7 (Completeness (cf. [AF22])). *An interactive proof $\Pi = (\mathsf{P}, \mathsf{V})$ for relation R is* complete *with completeness error $\rho : \{0,1\}^* \to [0,1]$ if for every $(x; w) \in R$,*

$$\Pr[(\mathsf{P}(w), \mathsf{V})(x) = \mathit{reject}] \leq \rho(x).$$

If $\rho(x) = 0$ for all $x \in L_R$, then Π is said to be perfectly *complete.*

Definition 2.8 (Soundness (cf. [AF22])). *An interactive protocol $\Pi = (\mathsf{P}, \mathsf{V})$ for relation R is* sound *with soundness error $\sigma : \{0,1\}^* \to [0,1]$ if for every $x \notin L_R$ and every prover P^*,*

$$\Pr[(\mathsf{P}^*, \mathsf{V})(x) = \mathit{accept}] \leq \sigma(x).$$

An interactive proof which is complete and sound allows a prover to convince a verifier that the statement x is true, i.e., $x \in L_R$. However, this does not necessarily convince a verifier that the prover actually "knows" the witness $w \in R(x)$. This stronger property is captured by the notion of *knowledge soundness*. Informally, knowledge soundness guarantees that if a prover convinces a verifier about the validity of some statement x with sufficiently high probability, then the prover can actually compute a witness $w \in R(x)$ with high probability. [1]

Definition 2.9 (Knowledge Soundness (cf. [AF22])). *An interactive protocol $\Pi = (\mathsf{P}, \mathsf{V})$ for relation R is* knowledge sound *with knowledge error $\varepsilon_{\mathsf{KS}} : \{0,1\}^* \to [0,1]$ if there exists a positive polynomial q and an algorithm Ext, called* knowledge extractor*, with the following properties: The extractor Ext, given input x and rewindable oracle access to a (potentially dishonest) prover P^*, runs in an expected number of steps that is polynomial in $|x|$ and outputs a witness $w \in R(x)$ with probability*

$$\Pr\left[\left(x; \mathsf{Ext}^{\mathsf{P}^*}(x)\right) \in R\right] \geq \frac{\epsilon(x, \mathsf{P}^*) - \varepsilon_{\mathsf{KS}}(x)}{\mathsf{q}(|x|)},$$

where $\epsilon(x, \mathsf{P}^*) := \Pr[(\mathsf{P}^*, \mathsf{V})(x) = \mathit{accept}]$.

If $\epsilon(x, \mathsf{P}^*) = \Pr[(\mathsf{P}^*, \mathsf{V})(x) = \mathsf{accept}] > \varepsilon_{\mathsf{KS}}(x)$, then the success probability of the knowledge extractor Ext in Definition 2.9 is positive. Therefore, $\epsilon(x, \mathsf{P}^*) > \varepsilon_{\mathsf{KS}}(x)$ implies that x admits a witness, i.e., $x \in L_R$. Hence, knowledge soundness implies soundness.

[1] Since the protocol presented in this work only achieves computational soundness, and is secure when the prover runs in *polynomial time*, technically our protocol is an *argument of knowledge*. However, we avoid this distinction for simplicity.

Definition 2.10 (Proof of Knowledge (cf. [AF22])). *An interactive proof that is both complete with completeness error $\rho(\cdot)$ and knowledge sound with knowledge error $\varepsilon_{\mathsf{KS}}(\cdot)$ is a* Proof of Knowledge *(PoK) if there exists a polynomial q such that $1 - \rho(x) \geq \varepsilon_{\mathsf{KS}}(x) + 1/\mathsf{q}(|x|)$ for all x.*

It is desirable to have simple verifiers which can send uniform random challenges to the prover, and efficiently verify the transcript.

Definition 2.11 (Public-Coin (cf. [AFK22])). *An interactive proof $\Pi = (\mathsf{P}, \mathsf{V})$ is* public-coin *if all of V's random choices are made public, i.e. are part of the transcript. The message $\mathsf{ch}_i \xleftarrow{\$} \mathcal{CH}_i$ of V in the $2i$-th round is called the i-th* challenge*, and $\mathcal{CH}_i$ is the* challenge set*.*

Public-coin protocols can be turned into non-interactive protocols by using the Fiat-Shamir transformation [FS87]. In this work, we consider only public-coin protocols.

Next, we discuss the notion of *special-soundness*. Special-soundness property is easier to check than knowledge soundness and for many protocols knowledge soundness follows from special-soundness. Note that this requires special-sound protocols to be public-coin.

Definition 2.12 (k-out-of-N Special Soundness (cf. [AF22])). *Let $k, N \in \mathbb{N}$. A 3-round public-coin protocol $\Pi = (\mathsf{P}, \mathsf{V})$ for relation R, with challenge set of cardinality $N \geq k$, is* k-out-of-N special sound *if there exists a polynomial time algorithm that, on input a statement x and k accepting transcripts $(\mathsf{cmt}, \mathsf{ch}_1, \mathsf{rsp}_1), \ldots, (\mathsf{cmt}, \mathsf{ch}_k, \mathsf{rsp}_k)$ with common first message cmt and pairwise distinct challenges $\mathsf{ch}_1, \ldots, \mathsf{ch}_k$, outputs a witness $w \in R(x)$. We also say Π is* k-special-sound *and, if $k = 2$, it is simply called special-sound.*

In order to generalize k-special-soundness to multi-round protocols we will introduce the notion of a tree of transcripts following the definitions given in [ACK21].

Definition 2.13 (Tree of Transcripts (cf. [AF22])). *Let $k_1, \ldots, k_\mu \in \mathbb{N}$. A $(k_1, \ldots, k_\mu)$-tree of transcripts for a $(2\mu + 1)$-round public-coin protocol $\Pi = (\mathsf{P}, \mathsf{V})$ is a set of $K = \prod_{i=1}^{\mu} k_i$ transcripts arranged in the following tree structure. The nodes in this tree correspond to the prover's messages and the edges to the verifier's challenges. Every node at depth i has precisely k_i children corresponding to k_i pairwise distinct challenges. Every transcript corresponds to exactly one path from the root to a leaf node. For a graphical representation we refer to Figure 1. We refer to the corresponding tree of challenges as a $(k_1, \ldots, k_\mu)$-tree of challenges.*

We will also write $\mathbf{k} = (k_1, \ldots, k_\mu) \in \mathbb{N}^\mu$ and refer to a $\mathbf{k}$-tree of transcripts.

Definition 2.14 ($(k_1, \ldots, k_\mu)$-out-of-$(N_1, \ldots, N_\mu)$ Special Soundness (cf. [AF22])). *Let $k_1, \ldots, k_\mu, N_1, \ldots, N_\mu \in \mathbb{N}$. A $(2\mu + 1)$-round public-coin protocol $\Pi = (\mathsf{P}, \mathsf{V})$ for a relation R, where V samples the i-th challenge from a set*

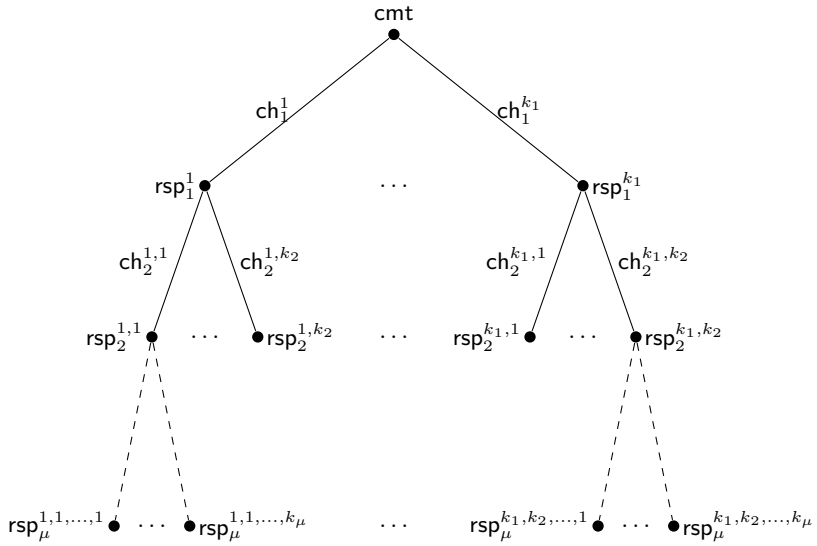


Fig. 1: $(k_1, k_2, \ldots, k_\mu)$ tree of transcripts of a $(2\mu+1)$-round public-coin protocol

of cardinality $N_i \geq k_i$ for $1 \leq i \leq \mu$, is $(k_1, \ldots, k_\mu)$-out-of-$(N_1, \ldots, N_\mu)$ special-sound if there exists a polynomial time algorithm that, on a input statement x and a $(k_1, \ldots, k_\mu)$-tree of accepting transcripts outputs a witness $w \in R(x)$. We also say Π is $(k_1, \ldots, k_\mu)$ special-sound.

The following theorem proved in [ACK21] states that special soundness implies knowledge soundness.

Theorem 2.1 ($(k_1, \ldots, k_\mu)$ Special Soundness implies Knowledge Soundness [ACK21, Theorem 1]). *Let $\mu, k_1, \ldots, k_\mu \in \mathbb{N}$ be such that $K = \Pi_{i=1}^{\mu} k_i$ can be upper bounded by a polynomial. Let (P, V) be a $(k_1, \ldots, k_\mu)$ special sound $(2\mu+1)$-round interactive protocol for relation R, where V samples each challenge uniformly at random from a set of cardinality N_i for $1 \leq i \leq \mu$. Then (P, V) is knowledge sound with knowledge error*

$$\varepsilon_{\mathsf{KS}} = \frac{\prod_{i=1}^{\mu} N_i - \prod_{i=1}^{\mu}(N_i - k_i + 1)}{\prod_{i=1}^{\mu} N_i} \leq \sum_{i=1}^{\mu} \frac{k_i - 1}{N_i} \tag{1}$$

We write $\Pi^\tau := (\mathsf{P}^\tau, \mathsf{V}^\tau)$ for the τ-fold parallel repetition of Π, which runs τ instances of Π in parallel and the verifier V^τ accepts if *all* the parallel instances are accepted.

The following theorem proved in [AF22] states that the knowledge soundness is retained (and knowledge error is reduced) via parallel repetition.

Theorem 2.2 (Parallel Repetition for Multi-Round Protocols [AF22, Theorem 4]). *Let (P, V) be a $(k_1, \ldots, k_\mu)$-out-of-$(N_1, \ldots, N_\mu)$ special sound*

protocol. Let $(\mathsf{P}^\tau, \mathsf{V}^\tau)$ *be the* τ*-fold repetition of protocol* (P, V) *. Then* $(\mathsf{P}^\tau, \mathsf{V}^\tau)$ *is knowledge sound with knowledge error* $\varepsilon_{\mathsf{KS}}^\tau$*, where*

$$\varepsilon_{\mathsf{KS}} = 1 - \prod_{i=1}^{\mu} \frac{(N_i - k_i + 1)}{N_i} \tag{2}$$

is the knowledge error of (P, V)*.*

Definition 2.15 (Special Honest-Verifier Zero Knowledge (SHVZK) (adapted from [ACK21])). *An interactive proof* $\Pi = (\mathsf{P}, \mathsf{V})$ *is called* { *perfectly, statistically, computationally* } honest-verifier zero knowledge (HVZK) *if there exists a polynomial time simulator that on input* $x \in L_R$ *outputs an accepting transcript which is distributed* { *perfectly, statistically, computationally* } *close to the transcripts generated by honest executions of* Π*. If the simulator proceeds by first sampling the verifier's messages uniformly at random, then* Π *is called* special honest-verifier zero knowledge (SHVZK)*.*

2.3.2 MPC-in-the-Head and PoK

Our construction relies on the *MPC-in-the-Head* (MPCitH) paradigm introduced by Ishai, Kushilevitz, Ostrovsky, and Sahai in [IKOS07, IKOS09]. This paradigm builds a zero-knowledge proof based on a secure multiparty computation (MPC) protocol. Informally, the MPC protocol is used to compute the verification of an NP relation, where the privacy guarantee of the protocol is used to achieve the zero-knowledge property.

The main steps of the proof of knowledge resulting from the MPCitH technique are the following:

1. The prover splits its witness into N parties by secret sharing the witness;
2. The prover then simulates locally ("in her head") all the parties of the MPC protocol which evaluates a boolean function that is expected to be 1 whenever the witness is correct (this is supposed to correspond to the verification of desired NP relation);
3. The prover commits to the views of all the parties in the MPC protocol;
4. The verifier chooses a random subset of $N' < N$ parties and asks to reveal their corresponding views;
5. The verifier finally checks that the views of the revealed parties are consistent with each other and with an honest execution of the MPC protocol that yields output 1.

This transformation achieves the zero-knowledge property as long as the views of any N' parties do not leak any information about the secret witness.

Since our proof of knowledge is an instantiation of the MPCitH technique for the specific case of $\mathsf{r\text{-}IPKP}$, it benefits from an extensive literature of optimizations generic to any MPCitH construction, such as:

– The preprocessing extension, introduced in [KKW18], allows the MPC protocol – used in the MPCitH technique – to rely on a preprocessing phase (under certain conditions) thus drastically reducing the proof size;
– the challenge space amplification technique, introduced in [BG23], that is itself an optimization of the PoK with Helper paradigm introduced in [Beu20];
– Merkle trees to reveal a partial number of random seeds, as explained in Section 2.2.4.

2.4 Digital Signature Schemes

2.4.1 Definition and properties

Definition 2.16 (Signature Scheme). *A signature scheme consists of three probabilistic polynomial time algorithms* $(\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Vf})$ *which work as follows:*

– $\mathsf{KeyGen}\left(1^\lambda\right)$*: The* key generation *algorithm takes a security parameter as input and outputs a pair of keys* $(\mathsf{pk}, \mathsf{sk})$*. The key* sk *is the private (secret)* signing key *and* pk *is the* public key *used for verification.*
– $\mathsf{Sign}_{\mathsf{sk}}(m)$*: The* signing *algorithm takes as input a secret signing key* sk *and a message* m *from some message space (that may depend on* pk*). It outputs a signature* $\sigma \leftarrow \mathsf{Sign}_{\mathsf{sk}}(m)$*.*
– $\mathsf{Vf}_{\mathsf{pk}}(m, \sigma)$*: The deterministic* verification *algorithm takes as input a public key* pk*, a message* m*, and a signature* σ*. It outputs a bit* $b := \mathsf{Vf}_{\mathsf{pk}}(m, \sigma)$*, with* $b = 1$ *meaning the signature-message pair is valid and* $b = 0$ *meaning it is invalid.*

Definition 2.17 (EUF-CMA **Security).** *A signature scheme* $(\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Vf})$ *is* EUF-CMA *secure if, for all* PPT *adversaries* $\mathcal{A}$ *there is a negligible function* $\mathsf{negl}(\cdot)$ *such that,*

$$\mathsf{Pr}\left[\begin{matrix}\mathsf{Vf}_{\mathsf{pk}}(m^*, \sigma^*) = 1 \bigwedge \\ (m^*, \cdot) \notin \mathcal{Q}^{\mathsf{Sign}}\end{matrix} \;\middle|\; \begin{matrix}(\mathsf{pk}, \mathsf{sk}) \longleftarrow \mathsf{KeyGen}(1^\lambda), \\ (m^*, \sigma^*) \longleftarrow \mathcal{A}^{\mathsf{Sign}_{\mathsf{sk}}(\cdot)}(\mathsf{pk})\end{matrix}\right] \leq \mathsf{negl}(\lambda).$$

where the environment keeps track of the queries to and from the signing oracle via $\mathcal{Q}^{\mathsf{Sign}}$*.*

2.4.2 Fiat-Shamir Transformation

In this section, we explain the random oracle model and Fiat-Shamir transformation used for transforming interactive protocols into non-interactive ones. We closely follow the presentation of [AFK22, Section 2.3] in the following exposition.

In the *random oracle model (ROM)*, algorithms have black-box (or input-output) access to an oracle $\mathsf{RO} : \{0,1\}^* \to \mathcal{Z}$, called as *random oracle*, which is instantiated with a uniform random function with domain $\{0,1\}^*$ and codomain $\mathcal{Z}$. Generally, $\mathcal{Z} = \{0,1\}^\eta$ for some $\eta \in \mathbb{N}$ related to the security parameter. In practice, RO can be implemented by lazy sampling, which means for each input

string $x \in \{0,1\}^*$, $\mathsf{RO}(x)$ is sampled uniform randomly from $\mathcal{Z}$ and then fixed. To avoid technical difficulties, we limit the domain from $\{0,1\}^*$ to $\{0,1\}^{\leq \ell}$, the finite set of all bitstrings of length at most ℓ, for a sufficiently large $\ell \in \mathbb{N}$.

An algorithm $\mathcal{A}^{\mathsf{RO}}$ that is given black-box access to a random oracle is called a *random oracle algorithm*. We say $\mathcal{A}$ is a *Q-query random-oracle algorithm*, if it makes at most Q queries to RO (independent of RO).

A natural extension of the ROM is when $\mathcal{A}$ is given access to *multiple independent* random oracles $\mathsf{RO}_1, \mathsf{RO}_2, \ldots, \mathsf{RO}_\mu$, possibly with different codomains. In practice, these random oracles can be instantiated by a single random oracle $\mathsf{RO} : \{0,1\}^* \to \{0,1\}^\eta$ using the standard techniques for domain separation (refer to [BDG20] for more details) and for sampling random elements from non-binary sets.

The Fiat-Shamir transform [FS87], turns a public-coin interactive proof into a non-interactive proof in random oracle model. The general idea of this transformation is to compute the i-th challenge message ch_i as a hash of the i-th prover message a_i along with (partial) communication transcript generated till that point. For a Σ-protocol, the challenge ch is computed as $\mathsf{ch} := \mathsf{H}(\mathsf{cmt})$ or as $\mathsf{ch} := \mathsf{H}(x, \mathsf{cmt})$, where the former is sufficient for *static* security, where the statement x is given as input to the dishonest prover, and the latter is necessary for *adaptive* security, where the dishonest prover can choose the statement x for which it wants to forge a proof.

For multi-round public-coin interactive proofs, there is some degree of freedom in the computation of the i-th challenge. For concreteness we consider a particular version where all previous messages are hashed along with the current message.

Let $\Pi = (\mathsf{P}, \mathsf{V})$ be a $(2\mu+1)$-round public-coin interactive proof, where the challenge for the i-th round is sampled from set $\mathcal{CH}_i$. For simplicity, we consider μ random oracles $\mathsf{RO}_i : \{0,1\}^{\leq \ell} \to \mathcal{CH}_i$ that map into the respective challenge spaces.

Definition 2.18 (Fiat-Shamir Transformation (cf. [AFK22])). *The static Fiat-Shamir transformation $\mathsf{FS}[\Pi] = (\mathsf{P}_{\mathsf{fs}}, \mathsf{V}_{\mathsf{fs}})$ is non-interactive proof in ROM, where $\mathsf{P}_{\mathsf{fs}}^{\mathsf{RO}_1.\mathsf{RO}_2....,\mathsf{RO}_\mu}(x; w)$ runs $\mathsf{P}(x; w)$ but instead of asking the verifier for the challenge ch_i on message a_i, the challenges are computed as*

$$\mathsf{ch}_i = \mathsf{RO}_i\left(a_1, a_2, \ldots, a_{i-1}, a_i\right); \tag{3}$$

the output is then the proof $\pi = (a_1, \ldots, a_{\mu+1})$. On input a statement x and a proof $\pi = (a_1, \ldots, a_{\mu+1})$, $\mathsf{P}_{\mathsf{fs}}^{\mathsf{RO}_1.\mathsf{RO}_2....,\mathsf{RO}_\mu}(x, \pi)$ accepts if, for ch_i as above V accepts the transcript $(a_1, \mathsf{ch}_1, \ldots a_\mu, \mathsf{ch}_\mu, a_{\mu+1})$ on input x.

If the challenges are computed as

$$\mathsf{ch}_i = \mathsf{RO}_i\left(x, a_1, \mathsf{ch}_1, \ldots a_{i-1}, \mathsf{ch}_{i-1}, a_i\right); \tag{4}$$

the resulting non-interactive proof in ROM is called as the adaptive *Fiat-Shamir transformation.*

2.5 The Permuted Kernel Problem and its Variants

In this subsection we give definitions of the computationally hard problems underlying the security of our proposed signature scheme.

We start by defining the classical Permuted Kernel Problem [Sha90] in its generic form, similar to [SBC22]: with an inhomogeneous syndrome $\mathbf{y}$ as well as a dimension parameter t.

Definition 2.19 (IPKP problem). *Let (q, m, n, t) be positive integers such that $m < n$, $\mathbf{H} \in \mathbb{F}_q^{m \times n}$, $(\mathbf{x}_i, \mathbf{y}_i) \in \mathbb{F}_q^n \times \mathbb{F}_q^m$ and $\pi \in \mathcal{S}_n$ be a permutation such that $\mathbf{H}(\pi[\mathbf{x}_i]) = \mathbf{y}_i$ for $i \in [t]$. Furthermore, the matrix whose columns are the $\mathbf{x}_i$ has full rank. Given $\left(\mathbf{H}, (\mathbf{x}_i, \mathbf{y}_i)_{i\in[t]}\right)$, the Inhomogeneous Permuted Kernel Problem $\mathsf{IPKP}(q, m, n, t)$ asks to find π.*

The IPKP problem was originally introduced with $t = 1$ only; in the rest of the article we refer to this version of the problem as *mono-dimensional* IPKP. Correspondingly, we refer with *multi-dimensional* IPKP to instances with arbitrary choices of $t > 1$.

Instead of directly relying on the hardness of IPKP, we consider a relaxed version r-IPKP which allows for more efficient constructions. In this relaxed variant the searched permutation does not necessarily have to satisfy the identity for all given pairs but only for an arbitrary (non-zero) linear combination of those pairs.

Definition 2.20 (r-IPKP). *Let (q, m, n, t) be positive integers such that $m < n$, $\mathbf{H} \in \mathbb{F}_q^{m \times n}$, $(\mathbf{x}_i, \mathbf{y}_i) \in \mathbb{F}_q^n \times \mathbb{F}_q^m$ and $\pi \in \mathcal{S}_n$ be a permutation such that $\mathbf{H}(\pi[\mathbf{x}_i]) = \mathbf{y}_i$ for $i \in [t]$. Furthermore, the matrix whose columns are the $\mathbf{x}_i$ has full rank. Given $\left(\mathbf{H}, (\mathbf{x}_i, \mathbf{y}_i)_{i\in[t]}\right)$, the Relaxed Inhomogeneous Permuted Kernel Problem $\mathsf{r\text{-}IPKP}(q, m, n, t)$ asks to find any $\tilde{\pi} \in \mathcal{S}_n$ such that $\mathbf{H}\left(\tilde{\pi}\left[\sum_{i\in[t]} \kappa_i \cdot \mathbf{x}_i\right]\right) = \sum_{i\in[t]} \kappa_i \cdot \mathbf{y}_i$ for any $\kappa \in \mathbb{F}_q^t \setminus \mathbf{0}$, where $\kappa := \{\kappa_1, \ldots, \kappa_t\}$ and $\mathbf{0} \in \mathbb{F}_q^t$ is the all zero vector.*

The respective hardness of the two above problems are discussed in Section 7.3.

3 Specifications

3.1 PoK for the r-IPKP Problem

Let $x = \left(\mathbf{H}, (\mathbf{x}_i, \mathbf{y}_i)_{i\in[t]}\right)$ and let $w = \pi \in \mathcal{S}_n$ as defined in Definition 2.20. Let $\mathcal{R}_{t-\mathsf{r\text{-}IPKP}}$ be a relation for r-IPKP problem defined as,

$$\mathcal{R}_{t-\mathsf{r\text{-}IPKP}} := \left\{ \left(\left(\mathbf{H}, (\mathbf{x}_i, \mathbf{y}_i)_{i\in[t]}\right); \tilde{\pi} \right) : \begin{array}{l} \mathbf{H}\left(\tilde{\pi}\left[\sum_{i\in[1,t]} \kappa_i \cdot \mathbf{x}_i\right]\right) = \sum_{i\in[1,t]} \kappa_i \cdot \mathbf{y}_i \\ \text{for any } \kappa \in {\mathbb{F}_q}^t \setminus \mathbf{0} \end{array} \right\}$$

We now present our protocol in Figure 2 that is inspired from [BG23] and [FJR23]. Informally, it consists of three main steps, following the MPCitH paradigm:

1. In the commitment step, the witness π is split into N *compositional* shares $\pi_1, \ldots, \pi_N$ such that $\pi = \pi_N \circ \pi_{N-1} \circ \cdots \circ \pi_1$. The prover also generates N (pseudo) random vectors $\mathbf{v}_1, \ldots, \mathbf{v}_N$ in $\mathbb{F}_q^n$. The compositional and vector shares are then combined to construct a syndrome $\mathbf{Hv}$, which is committed together with the generated shares (π_is and $\mathbf{v}_i$s).

2. The verifier then sends coefficients κ_i of an $\mathbb{F}_q$-linear combination as a first challenge. The prover then computes values $\mathbf{s}_1, \ldots, \mathbf{s}_N$ with the help of the π_i and $\mathbf{v}_i$ values committed earlier and the public statement $x = \big(\mathbf{H}, (\mathbf{x}_i, \mathbf{y}_i)_{i \in [t]}\big)$, such that $\mathbf{Hs}_N = \mathbf{Hv} + \sum_{i \in [t]} \kappa_i \mathbf{y}_i$. The prover then sends $\mathbf{s}_i$ values as its response. In the actual protocol we use a collision-resistant hash function to compress the information sent to the verifier.

3. Finally, the verifier sends an index $\alpha \in [N]$ as the second challenge. The prover reveals all shares π_i and $\mathbf{v}_i$ except the ones with index α. Additionally, the prover reveals the share $\mathbf{s}_\alpha$. This allows the verifier to verify the consistency of the views of all the shares except the ones with index α by recomputing the commitments. The verifier can also recompute all the $\mathbf{s}_i$ values for $i \neq \alpha$ and together with $\mathbf{s}_\alpha$ sent by the prover, the verifier can then reconstruct $\mathbf{s}_N$. Finally the verifier computes $\mathbf{Hv} = \mathbf{Hs}_N - \sum \kappa_i \mathbf{y}_i$ and checks if this value is consistent with the commitment received in first message (Step 1 above).

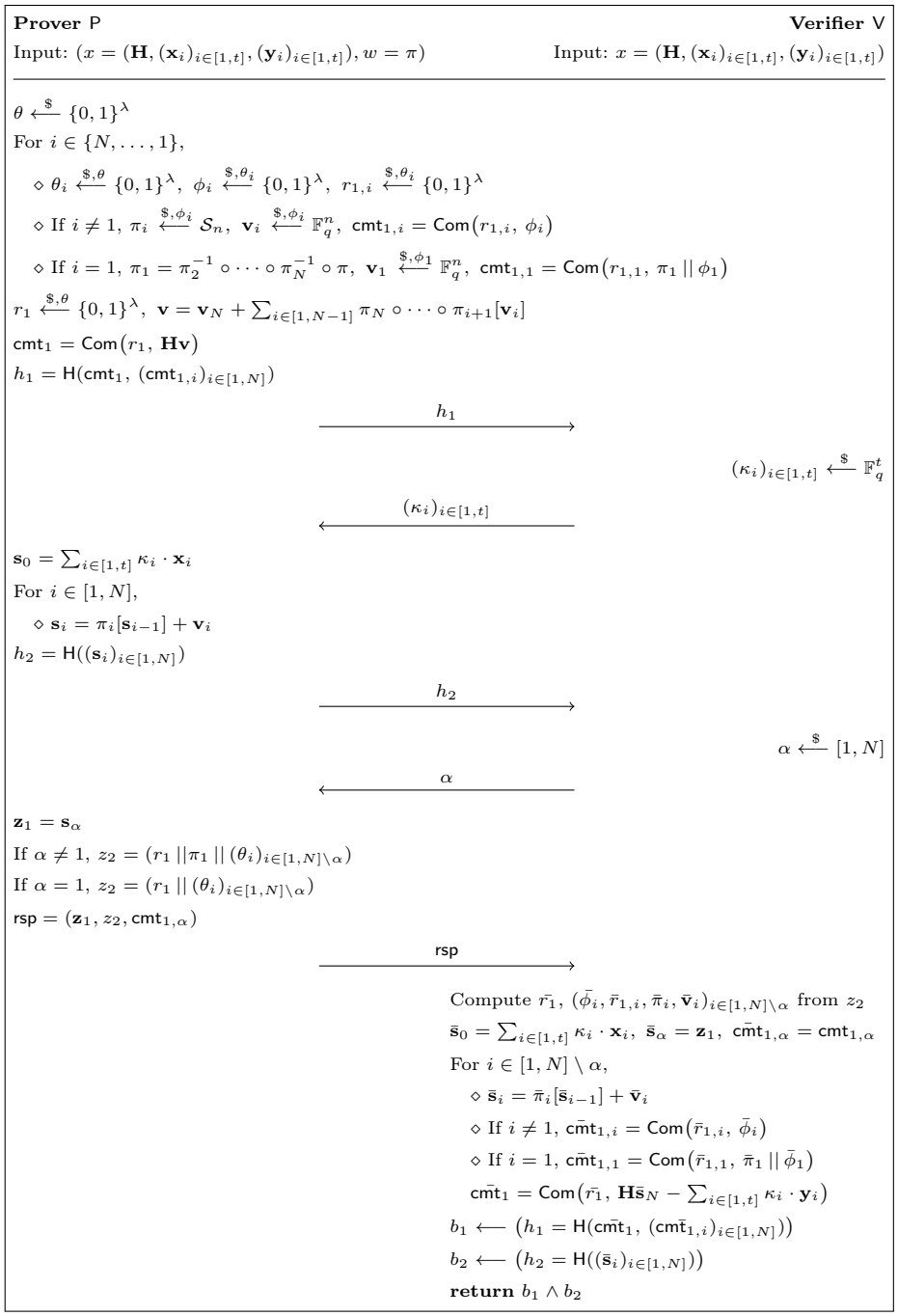


Fig. 2: PoK leveraging structure for the r-IPKP problem

The protocol in Figure 2 satisfies completeness, knowledge soundness, and zero-knowledge properties as stated in Theorem 3.1, Theorem 3.2, and Theorem 3.3 respectively.

Theorem 3.1 (Completeness). *The protocol presented in Figure 2 is* perfectly *complete.*

Proof. The completeness follows from the protocol description once it is observed that $\mathbf{s}_N = \pi\left[\sum_{i\in[1,t]} \kappa_i \cdot \mathbf{x}_i\right] + \mathbf{v}$ which implies that

$$\mathbf{H}\mathbf{s}_N - \sum_{i\in[1,t]} \kappa_i \cdot \mathbf{y}_i = \mathbf{H}\pi\left[\sum_{i\in[1,t]} \kappa_i \cdot \mathbf{x}_i\right] + \mathbf{H}\mathbf{v} - \sum_{i\in[1,t]} \kappa_i \cdot \mathbf{y}_i = \mathbf{H}\mathbf{v}.$$

Therefore for every true statement $(\mathbf{H}, (\mathbf{x}_i)_{i\in[1,t]}, (\mathbf{y}_i)_{i\in[1,t]})$ with witness π if the protocol described in Figure 2 is executed honestly then the verifier V accepts with probability 1 for all possible random choices of P and V. □

Theorem 3.2 (Knowledge Soundness). *The protocol presented in Figure 2 is knowledge sound with knowledge error*

$$\varepsilon_{\mathsf{KS}} = \frac{1}{N} + \frac{N-1}{N\cdot(q^t-1)}\,.$$

Theorem 3.3 (Special Honest-Verifier Zero Knowledge). *Assume that there exists a* $(\mathsf{t}, \epsilon_{\mathsf{PRG}})$*-secure* PRG*, and the commitment scheme* Com *is* $(\mathsf{t}, \epsilon_{\mathsf{Com}})$*-hiding. Then there exists an efficient simulator* Sim *which, outputs a transcript such that no distinguisher running in time at most* $\mathsf{t}(\lambda)$ *can distinguish between the transcript produced by* Sim *and a real transcript obtained by honest execution of the protocol in Figure 2 with probability better than* $(\epsilon_{\mathsf{PRG}}(\lambda) + \epsilon_{\mathsf{Com}}(\lambda))$*.*

We prove Theorem 3.2 and Theorem 3.3 in Section 6.1.1 and Section 6.1.2 respectively.

3.2 The PERK Digital Signature Scheme

We now present the signature scheme in Figure 3, Figure 4, and Figure 5 based on the Fiat-Shamir transformation of protocol shown in Figure 2.

1. Sample $\mathsf{sk\_seed} \xleftarrow{\$} \{0,1\}^\lambda$ and $\mathsf{pk\_seed} \xleftarrow{\$} \{0,1\}^\lambda$
2. Sample $\pi \longleftarrow \mathsf{PRG}(\mathsf{sk\_seed})$ from $\mathcal{S}_n$
3. Sample $(\mathbf{H}, (\mathbf{x}_j)_{j\in[1,t]}) \longleftarrow \mathsf{PRG}(\mathsf{pk\_seed})$ from $\mathbb{F}_q^{m\times n} \times (\mathbb{F}_q^n)^t$
3. For $j \in [1,t]$,
   - ◇ Compute $\mathbf{y}_j = \mathbf{H}\pi[\mathbf{x}_j]$
4. Output $(\mathsf{sk}, \mathsf{pk}) = (\mathsf{sk\_seed}, (\mathsf{pk\_seed}, (\mathbf{y}_j)_{j\in[1,t]}))$

Fig. 3: PERK - KeyGen algorithm

Inputs

- Secret key $\mathsf{sk} = \mathsf{sk\_seed}$
- Public key $\mathsf{pk} = (\mathsf{pk\_seed}, (\mathbf{y}_j)_{j\in[1,t]})$
- Message $m \in \{0,1\}^*$

Step 1: Commitment

1. Sample $\pi \longleftarrow \mathsf{PRG}(\mathsf{sk\_seed})$ from $\mathcal{S}_n$
2. Sample $(\mathbf{H}, (\mathbf{x}_j)_{j\in[1,t]}) \longleftarrow \mathsf{PRG}(\mathsf{pk\_seed})$ from $\mathbb{F}_q^{m\times n} \times (\mathbb{F}_q^n)^t$
3. Sample salt and master seed $(\mathsf{salt}, \mathsf{mseed}) \xleftarrow{\$} \{0,1\}^{2\lambda} \times \{0,1\}^{\lambda}$
4. Sample seeds $(\theta^{(e)})_{e\in[1,\tau]} \longleftarrow \mathsf{PRG}(\mathsf{salt}, \mathsf{mseed})$ from $(\{0,1\}^{\lambda})^{\tau}$
5. For each iteration $e \in [1,\tau]$,
   - $\diamond$ Compute $(\theta_i^{(e)})_{i\in[1,N]} \longleftarrow \mathsf{TreePRG}(\mathsf{salt}, \theta^{(e)})$
   - $\diamond$ For each party $i \in \{N,\dots,1\}$,
     - If $i \neq 1$, sample $(\pi_i^{(e)}, \mathbf{v}_i^{(e)}) \longleftarrow \mathsf{PRG}(\mathsf{salt}, \theta_i^{(e)})$ from $\mathcal{S}_n \times \mathbb{F}_q^n$
     - If $i = 1$, sample $\mathbf{v}_1^{(e)} \longleftarrow \mathsf{PRG}(\mathsf{salt}, \theta_1^{(e)})$ from $\mathbb{F}_q^n$
     - If $i \neq 1$, compute $\mathsf{cmt}_{1,i}^{(e)} = \mathsf{H}_0(\mathsf{salt}, e, i, \theta_i^{(e)})$
     - If $i = 1$, compute $\pi_1^{(e)} = (\pi_2^{(e)})^{-1} \circ \cdots \circ (\pi_N^{(e)})^{-1} \circ \pi$ and $\mathsf{cmt}_{1,1}^{(e)} = \mathsf{H}_0(\mathsf{salt}, e, 1, \pi_1^{(e)}, \theta_1^{(e)})$
   - $\diamond$ Compute $\mathbf{v}^{(e)} = \mathbf{v}_N^{(e)} + \sum_{i\in[1,N-1]} \pi_N^{(e)} \circ \cdots \circ \pi_{i+1}^{(e)}[\mathbf{v}_i^{(e)}]$ and $\mathsf{cmt}_1^{(e)} = \mathsf{H}_0(\mathsf{salt}, e, \mathbf{H}\mathbf{v}^{(e)})$

Step 2: First Challenge

6. Compute $h_1 = \mathsf{H}_1(\mathsf{salt}, m, \mathsf{pk}, (\mathsf{cmt}_1^{(e)}, \mathsf{cmt}_{1,i}^{(e)})_{e\in[1,\tau],i\in[1,N]})$
7. Sample $(\kappa_j^{(e)})_{e\in[1,\tau],j\in[1,t]} \longleftarrow \mathsf{PRG}(h_1)$ from $(\mathbb{F}_q^t)^{\tau}$

Step 3: First Response

8. For each iteration $e \in [1,\tau]$,
   - $\diamond$ Compute $\mathbf{s}_0^{(e)} = \sum_{j\in[1,t]} \kappa_j^{(e)} \cdot \mathbf{x}_j$
   - $\diamond$ For each party $i \in [1,N]$,
     - Compute $\mathbf{s}_i^{(e)} = \pi_i^{(e)}[\mathbf{s}_{i-1}^{(e)}] + \mathbf{v}_i^{(e)}$

Step 4: Second Challenge

9. Compute $h_2 = \mathsf{H}_2(\mathsf{salt}, m, \mathsf{pk}, h_1, (\mathbf{s}_i^{(e)})_{e\in[1,\tau],i\in[1,N]})$
10. Sample $(\alpha^{(e)})_{e\in[1,\tau]} \longleftarrow \mathsf{PRG}(h_2)$ from $([1,N])^{\tau}$

Step 5: Second Response

11. For each iteration $e \in [1,\tau]$,
    - $\diamond$ Compute $\mathbf{z}_1^{(e)} = \mathbf{s}_{\alpha}^{(e)}$
    - $\diamond$ If $\alpha^{(e)} \neq 1$, $z_2^{(e)} = (\pi_1^{(e)} \,||\, (\theta_i^{(e)})_{i\in[1,N]\setminus\alpha^{(e)}})$
    - $\diamond$ If $\alpha^{(e)} = 1$, $z_2^{(e)} = (\theta_i^{(e)})_{i\in[1,N]\setminus\alpha^{(e)}}$
    - $\diamond$ Compute $\mathsf{rsp}^{(e)} = (\mathbf{z}_1^{(e)}, z_2^{(e)}, \mathsf{cmt}_{1,\alpha^{(e)}}^{(e)})$
12. Compute $\sigma = (\mathsf{salt}, h_1, h_2, (\mathsf{rsp}^{(e)})_{e\in[1,\tau]})$

Fig. 4: PERK - Sign algorithm

Inputs

- Public key $\mathsf{pk} = (\mathsf{pk\_seed}, (\mathbf{y}_j)_{j\in[1,t]})$
- Signature σ
- Message $m \in \{0,1\}^*$

Step 1: Parse signature

1. Sample $(\mathbf{H}, (\mathbf{x}_j)_{j\in[1,t]}) \longleftarrow \mathsf{PRG}(\mathsf{pk\_seed})$ from $\mathbb{F}_q^{m\times n} \times (\mathbb{F}_q^n)^t$
2. Parse signature as $\sigma = (\mathsf{salt}, h_1, h_2, (\mathbf{z}_1^{(e)}, z_2^{(e)}, \mathsf{cmt}_{1,\alpha^{(e)}}^{(e)})_{e\in[1,\tau]})$
3. Recompute $(\kappa_j^{(e)})_{e\in[1,\tau],j\in[1,t]} \longleftarrow \mathsf{PRG}(h_1)$ from $(\mathbb{F}_q^t)^\tau$
4. Recompute $(\alpha^{(e)})_{e\in[1,\tau]} \longleftarrow \mathsf{PRG}(h_2)$ from $([1,N])^\tau$

Step 2: Verification

5. For each iteration $e \in [1,\tau]$,
   - ◇ Compute $\mathbf{s}_0^{(e)} = \sum_{j\in[1,t]} \kappa_j^{(e)} \cdot \mathbf{x}_j$ and $\mathbf{s}_\alpha^{(e)} = \mathbf{z}_1^{(e)}$
   - ◇ Compute $(\pi_i^{(e)}, \mathbf{v}_i^{(e)})_{i\in[1,N]\setminus\alpha}$ from $\mathbf{z}_2^{(e)}$
   - ◇ For each party $i \in [1,N] \setminus \alpha^{(e)}$,
     - If $i \neq 1$, compute $\mathsf{cmt}_{1,i}^{(e)} = \mathsf{H}_0(\mathsf{salt}, e, i, \theta_i^{(e)})$
     - If $i = 1$, compute $\mathsf{cmt}_{1,1}^{(e)} = \mathsf{H}_0(\mathsf{salt}, e, 1, \pi_1^{(e)}, \theta_1^{(e)})$
   - ◇ For each party $i \in [1,N] \setminus \alpha^{(e)}$,
     - Compute $\mathbf{s}_i^{(e)} = \pi_i^{(e)}[\mathbf{s}_{i-1}^{(e)}] + \mathbf{v}_i^{(e)}$
   - ◇ Compute $\mathsf{cmt}_1^{(e)} = \mathsf{H}_0(\mathsf{salt}, e, \mathbf{Hs}_N^{(e)} - \sum_{j\in[1,t]} \kappa_j^{(e)} \cdot \mathbf{y}_j)$
6. Compute $\bar{h}_1 = \mathsf{H}_1(\mathsf{salt}, m, \mathsf{pk}, (\mathsf{cmt}_1^{(e)}, \mathsf{cmt}_{1,i}^{(e)})_{e\in[1,\tau],i\in[1,N]})$.
7. Compute $\bar{h}_2 = \mathsf{H}_2(\mathsf{salt}, m, \mathsf{pk}, h_1, (\mathbf{s}_i^{(e)})_{e\in[1,\tau],i\in[1,N]})$.
8. Output accept if and only if $\bar{h}_1 = h_1$ and $\bar{h}_2 = h_2$.

Fig. 5: PERK - Verify algorithm

Theorem 3.4. *Suppose* PRG *is* $(\mathsf{t}, \epsilon_{\mathsf{PRG}})$*-secure and any adversary running in time* $\mathsf{t}(\lambda)$ *can solve the the underlying* $\mathsf{r\text{-}IPKP}$ *instance with probability at most* $\epsilon_{\mathsf{r\text{-}IPKP}}$*. Model* H_0*,* H_1*, and* H_2 *as random oracles where* H_0*,* H_1*, and* H_2 *have* 2λ*-bit output length. Then chosen-message attacker against the signature scheme (PERK) presented in Figure 4, running in time* $\mathsf{t}(\lambda)$*, making* q_s *signing queries, and making* q_0*,* q_1*,* q_2 *queries, respectively, to the random oracles, succeeds in outputting a valid forgery with probability*

$$\Pr[\mathsf{Forge}] \leq \frac{(q_0 + \tau\cdot(N+1)\cdot q_s)^2}{2\cdot 2^{2\lambda}} + \frac{q_s\cdot(q_0+q_1+q_2+q_s)}{2^{2\lambda}} + \tau\cdot q_s\cdot\epsilon_{\mathsf{PRG}}(\lambda) + \epsilon_{\mathsf{r\text{-}IPKP}} + q_2\cdot\varepsilon_{\mathsf{KS}}^{\tau}, \tag{5}$$

where $\varepsilon_{\mathsf{KS}} = \frac{1}{N} + \frac{N-1}{N\cdot(q^t-1)}$.

We prove Theorem 3.4 in Section 6.2.1.

4 Parameter Sets

4.1 Parameter Choice

The parameters were chosen as follows:

- *PKP parameters* (q, n, m, t): Parameters specific to the r-IPKP instance were chosen so as to minimize the signature size while offering concrete bit-security of r-IPKP above the NIST specified thresholds of 143, 207 and 272 bit for category I, III and V. Here we disregard any polynomial factors or additional cost induced by memory access. We give full details on estimating the security of r-IPKP in Section 7.3. We present two sets of parameters for each category, depending on whether $t = 3$ or $t = 5$.
- *MPC parameters* (N, τ): The number of parties and iterations is governed by the knowledge soundness of the protocol. The MPC parameters are also chosen to guarantee a soundness probability of $2^{-\lambda}$ for $\lambda \in \{128, 192, 256\}$ for category I, III and V respectively. For deriving the soundness we also take into account the Kales-Zavurecha attack, see Section 7.1. Following common practice we propose again two different parameter sets, a *short* variant using $N = 256$ and a fast variant using $N = 32$.

Table 1 presents our suggested parameter sets.

		PKP param.				MPC param.				
Parameter Set	λ	q	n	m	t	N	τ	pk size	sk size	σ size
PERK-I-fast3	128	1021	79	35	3	32	30	0.15 kB	16 B	8.35 kB
PERK-I-fast5	128	1021	83	36	5	32	28	0.24 kB	16 B	8.03 kB
PERK-I-short3	128	1021	79	35	3	256	20	0.15 kB	16 B	6.56 kB
PERK-I-short5	128	1021	83	36	5	256	18	0.24 kB	16 B	6.06 kB
PERK-III-fast3	192	1021	112	54	3	32	46	0.23 kB	24 B	18.8 kB
PERK-III-fast5	192	1021	116	55	5	32	43	0.37 kB	24 B	18.0 kB
PERK-III-short3	192	1021	112	54	3	256	31	0.23 kB	24 B	15.0 kB
PERK-III-short5	192	1021	116	55	5	256	28	0.37 kB	24 B	13.8 kB
PERK-V-fast3	256	1021	146	75	3	32	61	0.31 kB	32 B	33.3 kB
PERK-V-fast5	256	1021	150	76	5	32	57	0.51 kB	32 B	31.7 kB
PERK-V-short3	256	1021	146	75	3	256	41	0.31 kB	32 B	26.4 kB
PERK-V-short5	256	1021	150	76	5	256	37	0.51 kB	32 B	24.2 kB

Table 1: Parameters of PERK signature scheme. The aforementioned sizes are the ones used in our implementations except that we also concatenate the public key within the secret key in order to respect the NIST API.

4.2 Key and Signature Sizes

Key size. The private key as well as most of the components of the public key can be derived from a seed. The only elements not generated from a seed in the public key are the t syndromes $(\mathbf{y}_i)$.

Signature size. The signature consists of a salt and two hashes (h_1, h_2), making a subtotal of 6λ bits, and then τ repetitions of the following:

- A vector $\mathbf{z}_1^{(e)} \in \mathbb{F}_q^n$;
- A permutation in $\mathcal{S}_n$;
- $N-1$ seeds (of size λ) arranged in a PRG tree, hence of size only $\lambda \cdot \lceil \log_2(N) \rceil$;
- A commitment $\mathsf{cmt}_{1,\alpha^{(e)}}^{(e)}$ of size 2λ.

Overall, for a security level λ, the key and signature sizes for our signature scheme are captured by the following formulas:

Public key size (bits)
$\lambda + t \cdot m\lceil \log_2(q) \rceil$

Signature size (bits)
$6\lambda + \tau \cdot \Big(\underbrace{n\lceil \log_2(q) \rceil}_{\text{vector in } \mathbb{F}_q^n} + \underbrace{n\lceil \log_2(n) \rceil}_{\text{permutation}} + \underbrace{\lambda\lceil \log_2(N) \rceil}_{\text{seeds}} + \underbrace{2\lambda}_{\text{commitment}} \Big)$

Table 2: Public key and signature sizes in bits

Signature compression. Our implementation features an optimization that further reduce the aforementioned signature theoretical size. The idea is to pack the permutation elements two by two. Instead of representing a permutation $\pi \in \mathcal{S}_n$ as a sequence of n elements in $[0, n-1]$ it is represented as a sequence of $\lceil n/2 \rceil$ elements in $[0, n^2-1]$. When the following inequality holds

$$\left\lceil \log_2(n^2) \right\rceil < 2\lceil \log_2(n) \rceil,$$

the packed permutation takes less memory size. Numbers given in Table 1 take into account this compression technique.

5 Implementations and Performances

5.1 Instantiation of the scheme

5.1.1 Representation of objects

Field elements. Elements of $\mathbb{F}_q$ are stored in 16 bit unsigned integers.

Vectors. A vector of $\mathbb{F}_q^n$ (respectively $\mathbb{F}_q^m$) are represented as an array of length n (respectively of length m) of $\mathbb{F}_q$ elements.

Matrices. A matrix $\mathbf{H} \in \mathbb{F}_q^{m \times n}$ is represented as a two dimensional array of $\mathbb{F}_q$ elements i.e. an array of length m of arrays of length n.

Permutations. An element of $\mathcal{S}_n$ is represented as an array of length n of elements in $[0, n-1]$.

5.1.2 Randomness and objects generation

In order to sample elements of $\mathbb{F}_q$, we sample a random 16 bits value by getting two bytes from the PRG and assigning the first to the less significant bits and the second to the most significant bits (little endian), then we take the lower $\lceil \log_2(q) \rceil$ bits, rejecting the value if it is equal or greater than q. Random vectors in $\mathbb{F}_q^n$ (respectively matrices in $\mathbb{F}_q^{m \times n}$) are sampled uniformly by sampling n (respectively $m \times n$) elements in $\mathbb{F}_q$ following the aforementioned procedure.

Random permutations in S_n are generated using the constant time sorting software `djbsort` [Ber19]. More precisely, to generate a permutation $\pi \in S_n$, we start with a vector $\mathbf{v} = (v_0, \cdots, v_{n-1}) = (0, 1, \cdots, n-1)$, then we sample a random vector $\mathbf{e} = (e_0, \cdots, e_{n-1}) \in (\mathbb{F}_2^{16})^n$, and we construct a vector $\mathbf{p} = (p_0, \cdots, p_{n-1})$, where the high-order bits of p_i corresponds to e_i and the lower-bits order of p_i corresponds to v_i. We then sort this integer sequence in constant-time using `djbsort`, and we extract the permutation π from the lower-order bits $\mathbf{p}$. If there are any duplicate values in the vector $\mathbf{e}$, we discard it and generate a new one. To apply a permutation $\pi \in S_n$ to a vector $\mathbf{v} \in \mathbb{F}_q^n$, we follow the same process while replacing the vector $\mathbf{e}$ by the coefficients of the permutation π.

5.1.3 Parsing objects from/to byte strings

Vectors of $\mathbb{F}_q^n$ (respectively, $\mathbb{F}_q^m$) are converted to byte strings using a compact representation in which the unused bits of each element are removed, thus leading to $n\lceil \log_2(q) \rceil$ (respectively, $m\lceil \log_2(q) \rceil$) long bit string. The compact representation is used for the public key pk, to parse the t syndromes $(\mathbf{y}_i)$ and for the signature σ to parse the vectors $\mathbf{z}_1^{(e)} \in \mathbb{F}_q^n$. During the parsing, the coefficients of all the vectors are packed together ensuring that no space is lost between them.

To optimize space usage, the permutations coefficients are packed in couples using the following process. Let $A = \lfloor 2^b \rfloor$, where $b = 6.5$ for $\lambda = 128$ ($b = 7$ for $\lambda = 192$ and $b = 7.5$ for $\lambda = 256$). Let $\pi \in S_n$, then for any two consecutive coefficients c_a and c_b of π, we set $cp = (A * c_a) + c_b$. Then, the compact cp representations ($2 \cdot b$ bits) are concatenated and stored into a byte string. The coefficient are then unpacked as follows. We compute $c_a = \lfloor \frac{cp}{A} \rfloor$ and $c_b = cp \bmod A$. Its worth mentioning that for $\lambda = 192$ (i.e, $b = 7$), the packed coefficients are simply the concatenation $c_a || c_b$. During the parsing the all the cp of all permutations are packed together ensuring that no space is lost between them.

Objects are always stored in little endian representation.

5.1.4 PRG, Hash functions and TreePRG

PRG. The `randombytes` function provided by the NIST is used to sample uniformly at random the salt and various seeds ($\mathsf{sk\_seed}, \mathsf{pk\_seed}, \mathsf{mseed}$). The PRG function is instantiated using `SHAKE-128` for $\lambda = 128$ and `SHAKE-256` otherwise, along with domain separators.

Hash functions. The hash functions are instantiated using `SHA3-256` for $\lambda = 128$, `SHA3-384` for $\lambda = 192$ and `SHA3-512` for $\lambda = 256$ along with domain separators.

TreePRG. When signing, the signer must generate, for each iteration, a set of N seeds, then reveal $N - 1$ of these based on the challenge. The seeds are then used by the verifier to check that the MPC protocol was setup or simulated correctly. By deriving seeds deterministically in a binary tree, then using the leaf seeds in the protocol, the signer can reveal the $N - 1$ seeds efficiently by revealing intermediate nodes in the tree. In our setting, N is always a power of 2. Seeds are of size λ bits. Nodes and leaves, of size λ bits, are generated using the hash function. The constant $\mathtt{H}_3 = \mathtt{0x03}$ is a domain separator and encoded over one byte. The salt value salt is of size $2 \cdot \lambda$ bits.

Expand Tree (`sig_perk_expand_theta_tree`)
Input: A root seed $rseed$, a salt value salt.
Output: A binary tree consisting in a root seed, $N - 2$ nodes and N leaves (seeds).

Nodes, starting from $rseed$ ($node_0$), are indexed from top to down, left to right. Each node generates a digest ($2 \cdot \lambda$ bits wide) using the hash function H:

$$digest = \mathsf{H}\left(\mathsf{salt} || idx || node_i || \mathtt{H}_3\right)$$

where idx is one byte containing the node index. Then set the left child of the node to the leftmost λ bits, and the right child to the rightmost λ bits.

Get Partial Tree Seeds (`sig_perk_get_theta_partial_tree_seeds`)

Input: A binary tree consisting in a root seed, $N-2$ nodes and N leaves (seeds) and the index α of the hidden seed.
Output: An array of $L = \log_2(N)$ seeds that allow to rebuild the leaf seeds minus the α one.

The tree is traversed from top to bottom following the path to the α leaf. For each level after the root one, the sibling node/leaf is saved.

Expand Partial Tree (`sig_perk_expand_theta_partial_tree`)
Input: An array of $L = \log_2(N)$ seeds and the index α of the hidden seed.
Output: The partial seed tree where the nodes belonging to the path to α leaf are missing.

The tree is built as in **Expand Tree**, using the nodes in the array for the siblings belonging to the path to the α leaf.

5.1.5 Keys and signature representation The secret key $\mathsf{sk} = \pi$ is represented as $\mathsf{sk} = (\mathsf{sk\_seed})$ where $\mathsf{sk\_seed}$ is a λ bits seed used to generate π. The public key $\mathsf{pk} = (\mathbf{H}, (\mathbf{x}_i)_{i\in[1,t]}, (\mathbf{y}_i)_{i\in[1,t]})$ is represented as $\mathsf{pk} = (\mathsf{pk\_seed}, (\mathbf{y}_i)_{i\in[1,t]})$ where $\mathsf{pk\_seed}$ is a λ bits seed used to generate $\mathbf{H}$ and $(\mathbf{x}_i)_{i\in[1,t]}$. The signature σ is represented as $(\mathsf{salt}, h_1, h_2, (\mathsf{rsp}^{(e)})_{e\in[1,\tau]})$ where:

- salt is $2\cdot\lambda$ bits salt generated using the `randombytes` function ;
- h_1 (respectively, h_2) is a $2\cdot\lambda$ bits digest generated using the hash function $\mathsf{H}(\cdot\|\mathtt{H}_1)$ (respectively, $\mathsf{H}(\cdot\|\mathtt{H}_2)$). The constants $\mathtt{H}_1 = \mathtt{0x01}$ and $\mathtt{H}_2 = \mathtt{0x02}$ are domain separators and encoded over one byte ;
- $(\mathsf{rsp}^{(e)})_{e\in[1,\tau]}$ is a set of τ responses $\mathsf{rsp}^{(e)} = (\mathbf{z}_1^{(e)}, z_2^{(e)}, \mathsf{cmt}^{(e)}_{1,\alpha^{(e)}})$, where $\mathsf{cmt}^{(e)}_{1,\alpha^{(e)}}$ is $2\cdot\lambda$ bits commitment generated using the hash function $\mathsf{H}(\cdot\|\mathtt{H}_0)$. The constant $\mathtt{H}_0 = \mathtt{0x00}$ is a domain separator and encoded over one byte.

5.2 Benchmarks and Performances

This section provides performance measures of our PERK implementations.

Benchmark platform. The benchmarks have been performed on a machine running Ubuntu 22.04.2 LTS, that has 64 GB of memory and an Intel® Core™ i9-13900K @ 3.00 GHz for which the Hyper-Threading and Turbo Boost features were disabled. For each parameter set, the results have been obtained by computing the average from 1000 random instances. The scheme have been compiled with `gcc` (version `11.3.0`). The following third party libraries have been used: `XKCP` (commit `7fa59c0ec4`) and `djbsort` (version `20190516`).

The current implementation require more than 8 MBytes of stack to run the NIST level V parameter sets hence one might need to increase the available stack to run them on some platforms, see the technical documentation (readme file) for additional instructions.

5.2.1 Reference Implementation The reference implementation is written in `C` and have been compiled using the compilation flags `-O3 -funroll-all-loops -march=native`. The performances of our reference implementation on the aforementioned benchmark platform are described Tab. 3.

Parameter Set	Keygen	Sign	Verify
PERK-I-fast3	79	21	9.1
PERK-I-fast5	96	21	8.9
PERK-I-short3	81	114	47
PERK-I-short5	98	109	45
PERK-III-fast3	159	51	22
PERK-III-fast5	184	49	21
PERK-III-short3	172	273	116
PERK-III-short5	203	258	107
PERK-V-fast3	267	101	45
PERK-V-fast5	296	98	43
PERK-V-short3	278	548	245
PERK-V-short5	320	511	225

Table 3: Performances of the reference implementation for different instances of PERK. The key generation numbers are in kilo CPU cycles, while the signing and verification numbers are in million CPU cycles.

5.2.2 Optimized Implementation A constant-time optimized implementation leveraging `AVX2` instructions have been provided. Its performances on the aforementioned benchmark platform are described in Tab. 4. The following optimization flags have been used during compilation: `-O3 -funroll-all-loops -march=native -mavx2 -mpclmul -msse4.2 -maes`. There are two main differences between the reference and optimized implementation. Firstly, the optimized implementation utilizes optimized `AVX2` implementations provided by the libraries `XKCP` and `djbsort`, whereas the reference implementation relies on 64-bit platform plain `C` implementation of the aforementioned libraries. Secondly, the optimized implementation uses the vectorized implementation from `XKCP` of 4 parallel instances of Keccak, while the reference implementation uses the non parallel feature.

5.3 Known Answer Test Values

Known Answer Test (KAT) values have been generated using the script provided by the NIST. They are available in the folder `KATs` and files are the same for both

Parameter Set	Keygen	Sign	Verify
PERK-I-fast3	77	7.6	5.3
PERK-I-fast5	90	7.2	5.1
PERK-I-short3	80	39	27
PERK-I-short5	91	36	25
PERK-III-fast3	167	16	13
PERK-III-fast5	185	15	12
PERK-III-short3	175	82	65
PERK-III-short5	194	77	60
PERK-V-fast3	304	36	28
PERK-V-fast5	324	34	26
PERK-V-short3	300	185	143
PERK-V-short5	328	171	131

Table 4: Performances of the optimized implementation for different instances of PERK. The key generation numbers are in kilo CPU cycles, while the signing and verification numbers are in million CPU cycles.

reference and optimized implementation. In addition, examples with intermediate values have also been provided in these folders. Notice that one can generate the aforementioned test files using respectively the `kat` and `verbose` modes of our implementation. The procedure to follow in order to do so is detailed in the technical documentation.

6 Expected Security Strength

Our signature scheme is EUF-CMA in the ROM under the assumption of hardness of r-IPKP and with a choice of repetitions τ that makes the probability of a forgery negligible.

The proof of EUF-CMA, written below, happens in two stages:

- We first prove the knowledge soundness and the special honest-verifier zero knowledge of the PoK in Figure 2;
- We build on the previous proof to prove that the signature scheme is EUF-CMA.

The estimation of hardness for r-IPKP is given in 7.3 and the choice of τ is covered in 7.1.

6.1 Security Proofs for the Proof of Knowledge

6.1.1 Proof of Theorem 3.2 We restate the Theorem 3.2 below and follow it by its proof.

Theorem 3.2 (Knowledge Soundness). *The protocol presented in Figure 2 is knowledge sound with knowledge error*

$$\varepsilon_{\mathsf{KS}} = \frac{1}{N} + \frac{N-1}{N \cdot (q^t - 1)} .$$

Proof (Proof of Theorem 3.2). Before proving the knowledge soundness of our protocol, we will first prove the following useful lemma.

Lemma 6.1 ($(2,2)$-special soundness). *The protocol shown in Figure 2 is* $(2,2)$-special sound.

Proof (Proof of Lemma 6.1). $(2,2)$**-special soundness.** Following Definition 2.14 the protocol is called $(2,2)$-special sound if there exists an efficient knowledge extractor Ext which on an input statement $(\mathbf{H}, (\mathbf{x}_i)_{i\in[1,t]}, (\mathbf{y}_i)_{i\in[1,t]})$ and a $(2,2)$-accepting tree of transcripts (See Definition 2.13) returns a solution of the r-IPKP instance defined by $(\mathbf{H}, (\mathbf{x}_i)_{i\in[1,t]}, (\mathbf{y}_i)_{i\in[1,t]})$. We now show such an extractor which takes 4 accepting transcripts associated with challenges $(\kappa, \alpha_1), (\kappa, \alpha_2)$, $(\kappa', \alpha_1), (\kappa', \alpha_2)$ such that $\kappa = (\kappa_i)_{i\in[1,t]}$, $\kappa' = (\kappa'_i)_{i\in[1,t]}$ as well as $\kappa \neq \kappa'$ and $\alpha_1 \neq \alpha_2$, and outputs a solution to the r-IPKP instance defined by $(\mathbf{H}, (\mathbf{x}_i)_{i\in[1,t]}, (\mathbf{y}_i)_{i\in[1,t]})$.

Let $z_2^{(\kappa^*,\alpha^*)}$ denote the response z_2 computed as shown in Figure 2 when the first and second challenges are κ^* and α^* respectively. Note that, $z_2^{(\kappa,\alpha_1)}$ contains all the seeds θ_i for $i \in [1,N]$ except $i = \alpha_1$. Therefore, the extractor has access to all the seeds θ_i for $i \in [1,N]$ since it knows both $z_2^{(\kappa,\alpha_1)}$ as well as $z_2^{(\kappa,\alpha_2)}$ and $\alpha_1 \neq \alpha_2$. It can compute $(\bar{\pi}_i^{(\kappa)}, \bar{\mathbf{v}}_i^{(\kappa)})_{i\in[1,N]}$ and $(\bar{\pi}_i^{(\kappa')}, \bar{\mathbf{v}}_i^{(\kappa')})_{i\in[1,N]}$ from $\left(z_2^{(\kappa,\alpha_i)}\right)_{i\in[1,2]}$ and $\left(z_2^{(\kappa',\alpha_i)}\right)_{i\in[1,2]}$ respectively.

Also, note that the first message $h_1 = \mathsf{H}(\mathsf{cmt}_1, (\mathsf{cmt}_{1,i})_{i\in[1,N]})$ is common to all the 4 transcripts. Since we assume that H is a collision-resistant hash function, it means that the initial commitments $(\mathsf{cmt}_1, (\mathsf{cmt}_{1,i})_{i\in[1,N]})$ are all same in the 4 transcripts. From the binding property of the commitments $(\mathsf{cmt}_{1,i})_{i\in[1,N]}$, we know that

$$(\bar{\pi}_i, \bar{\mathbf{v}}_i)_{i\in[1,N]} = (\bar{\pi}_i^{(\kappa)}, \bar{\mathbf{v}}_i^{(\kappa)})_{i\in[1,N]} = (\bar{\pi}_i^{(\kappa')}, \bar{\mathbf{v}}_i^{(\kappa')})_{i\in[1,N]}.$$

The knowledge extractor Ext computes the solution as

1. Compute $(\bar{\pi}_i)_{i\in[1,n]}$ from $z_2^{(\kappa,\alpha_1)}$ and $z_2^{(\kappa,\alpha_2)}$
2. Output $\bar{\pi} = \bar{\pi}_N \circ \cdots \circ \bar{\pi}_1$

Let us now check the validity of this solution output by the extractor. By construction, we know that $\bar{\mathbf{s}}_0^{(\kappa,\alpha_1)} = \bar{\mathbf{s}}_0^{(\kappa,\alpha_2)} = \sum_{i\in[1,t]} \kappa_i \cdot \mathbf{x}_i$. Also, for all $i \in [1,N] \setminus \alpha_1$, $\bar{\mathbf{s}}_i^{(\kappa,\alpha_1)} = \bar{\pi}_i[\bar{\mathbf{s}}_{i-1}^{(\kappa,\alpha_1)}] + \bar{\mathbf{v}}_i$. And for all $i \in [1,N] \setminus \alpha_2$, $\bar{\mathbf{s}}_i^{(\kappa,\alpha_2)} = \bar{\pi}_i[\bar{\mathbf{s}}_{i-1}^{(\kappa,\alpha_2)}] + \bar{\mathbf{v}}_i$. Since the transcripts are accepting and V checks h_2 computed as $h_2 = \mathsf{H}((\mathbf{s}_i)_{i\in[1,N]})$, due to the collision-resistance property of H, it follows that for all $i \in [1,N]$, $\bar{\mathbf{s}}_i^{(\kappa)} = \bar{\pi}_i[\bar{\mathbf{s}}_{i-1}^{(\kappa)}] + \bar{\mathbf{v}}_i$, this implies $\bar{\mathbf{s}}_N^{(\kappa)} = \bar{\pi}[\sum_{i\in[1,t]} \kappa_i \cdot \mathbf{x}_i] + \bar{\mathbf{v}}$.

Following a similar argument, we know that $\bar{\mathbf{s}}_N^{(\kappa')} = \bar{\pi}[\sum_{i\in[1,t]} \kappa'_i \cdot \mathbf{x}_i] + \bar{\mathbf{v}}$. Based on the binding property of commitment cmt_1 and using the fact that the transcripts are accepting, we can write

$$\mathbf{H}\bar{\mathbf{s}}_N^{(\kappa)} - \sum_{i\in[1,t]} \kappa_i \cdot \mathbf{y}_i = \mathbf{H}\bar{\mathbf{s}}_N^{(\kappa')} - \sum_{i\in[1,t]} \kappa'_i \cdot \mathbf{y}_i$$

$$\implies \mathbf{H}(\bar{\pi}[\sum_{i\in[1,t]} \kappa_i \cdot \mathbf{x}_i] + \bar{\mathbf{v}}) - \sum_{i\in[1,t]} \kappa_i \cdot \mathbf{y}_i = \mathbf{H}(\bar{\pi}[\sum_{i\in[1,t]} \kappa'_i \cdot \mathbf{x}_i] + \bar{\mathbf{v}}) - \sum_{i\in[1,t]} \kappa'_i \cdot \mathbf{y}_i$$

$$\implies \mathbf{H}\bar{\pi}[\sum_{i\in[1,t]} (\kappa_i - \kappa'_i) \cdot \mathbf{x}_i] = \sum_{i\in[1,t]} (\kappa_i - \kappa'_i) \cdot \mathbf{y}_i$$

This implies that $\bar{\pi}$ is a solution of the considered r-IPKP problem. □

We can now apply the result of Theorem 2.1 to Lemma 6.1, which concludes the proof. □

6.1.2 Proof of Theorem 3.3 The following proof is inspired from the proof of [CDG+20, Lemma 6.1] and [FJR22a, FJR22b, Theorem 3]. We now show that the protocol described in Figure 2, Section 3.1, satisfies the special honest-verifier zero knowledge property. We assume that the commitment algorithm $\mathsf{Com}(\cdot)$ outputs $\ell(\lambda)$-bit strings as output for some polynomial ℓ. We restate the Theorem 3.3 below and follow it by its proof.

Theorem 3.3 (Special Honest-Verifier Zero Knowledge). *Assume that there exists a* $(\mathsf{t}, \epsilon_{\mathsf{PRG}})$*-secure* PRG*, and the commitment scheme* Com *is* $(\mathsf{t}, \epsilon_{\mathsf{Com}})$*-hiding. Then there exists an efficient simulator* Sim *which, outputs a transcript such that no distinguisher running in time at most* $\mathsf{t}(\lambda)$ *can distinguish between the transcript produced by* Sim *and a real transcript obtained by honest execution of the protocol in Figure 2 with probability better than* $(\epsilon_{\mathsf{PRG}}(\lambda) + \epsilon_{\mathsf{Com}}(\lambda))$*.*

Proof (Proof of Theorem 3.3). We begin by describing an efficient simulator Sim which outputs a transcript which is indistinguishable from a real transcript obtained by honest execution of the protocol. Sim on input $x = (\mathbf{H}, (\mathbf{x}_i)_{i\in[1,t]}, (\mathbf{y}_i)_{i\in[1,t]})$ works as follows:

1. Sample $\kappa \xleftarrow{\$} \mathbb{F}_q^t \setminus \mathbf{0}$ and $\alpha^* \xleftarrow{\$} [N]$
2. Sample $\theta \xleftarrow{\$} \{0,1\}^\lambda$
3. For each $i \in [1, N] \setminus \{\alpha^*\}$
   - $\diamond\ \theta_i \xleftarrow{\$,\theta} \{0,1\}^\lambda,\ \phi_i \xleftarrow{\$,\theta_i} \{0,1\}^\lambda,\ r_{1,i} \xleftarrow{\$,\theta_i} \{0,1\}^\lambda$
   - $\diamond$ **if** $i \neq 1$
     - $\triangleright\ \pi_i \xleftarrow{\$,\phi_i} \mathcal{S}_n,\ \mathbf{v}_i \xleftarrow{\$,\phi_i} \mathbb{F}_q^n,\ \mathsf{cmt}_{1,i} = \mathsf{Com}\big(r_{1,i},\ \phi_i\big)$
   - $\diamond$ **if** $i = 1$
     - $\triangleright\ \pi_1 \xleftarrow{\$} S_n,\ \mathbf{v}_1 \xleftarrow{\$,\phi_1} \mathbb{F}_q^n,\ \mathsf{cmt}_{1,1} = \mathsf{Com}\big(r_{1,1},\ \pi_1 \,||\, \phi_1\big)$
4. For $i = \alpha^*$
   - $\diamond$ Sample $\pi_{\alpha^*} \xleftarrow{\$} \mathcal{S}_n,\ \mathbf{v}_{\alpha^*} \xleftarrow{\$} \mathbb{F}_q^n,\ \mathsf{cmt}_{1,\alpha^*} \xleftarrow{\$} \{0,1\}^{\ell(\lambda)}$
5. Compute $r_1,\ \mathbf{v},\ \mathsf{cmt}_1,\ h_1$ as in the real protocol using the values computed above.
6. Compute $\tilde{\pi} = \pi_N \circ \cdots \circ \pi_1$
7. Compute $\tilde{\mathbf{x}}$ such that $\mathbf{H}\tilde{\mathbf{x}} = \sum_{i \in [1,t]} \kappa_i \cdot \mathbf{y}_i$
8. Compute $\mathbf{s}_0 = \sum_{i \in [1,t]} \kappa_i \cdot \mathbf{x}_i$
9. For each $i \in [1, \alpha^* - 1]$
   - $\diamond$ Compute $\mathbf{s}_i = \pi_i[\mathbf{s}_{i-1}] + \mathbf{v}_i$
10. Compute $\mathbf{s}_{\alpha^*} = \pi_{\alpha^*}[\mathbf{s}_{\alpha^*-1}] + \mathbf{v}_{\alpha^*} + \pi_{\alpha^*+1}^{-1} \circ \cdots \circ \pi_N^{-1}[\tilde{\mathbf{x}} - \tilde{\pi}[\sum_{i \in [1,t]} \kappa_i \cdot \mathbf{x}_i]]$
11. For each $i \in [\alpha^* + 1, N]$
    - $\diamond$ Compute $\mathbf{s}_i = \pi_i[\mathbf{s}_{i-1}] + \mathbf{v}_i$
12. Compute $h_2,\ z_1,\ z_2,\ \mathsf{rsp}$ as in the real protocol using the values computed above.
13. Output $(x, h_1, (\kappa_i)_{i \in [1,t]}, h_2, \alpha^*, \mathsf{rsp})$

Note that the simulator Sim runs in polynomial-time and the challenges sampled in Step 1 are distributed identically to the real world execution since the verifier also samples the challenges uniformly at random. We now show that the transcript output by Sim and a real transcript obtained by honest execution of the protocol in Figure 2 with challenges (κ, α^*) cannot be distinguished with probability better than $(\epsilon_{\mathsf{PRG}}(\lambda) + \epsilon_{\mathsf{Com}}(\lambda))$ by any distinguisher running in time at most $\mathsf{t}(\lambda)$. We consider the following sequence of simulators:

Simulator 0 (real world). This simulator takes the statement $x = (\mathbf{H}, (\mathbf{x}_i)_{i \in [1,t]}, (\mathbf{y}_i)_{i \in [1,t]})$, witness π, and the challenges (κ, α^*) as input. It then runs the protocol in Figure 2 honestly and outputs the transcript. This transcript is identically distributed as a real-world transcript.

Simulator 1. Simulator 1 works exactly same as Simulator 0 except that instead of computing $\mathsf{cmt}_{1,\alpha^*}$ as in the real protocol, it samples a uniform string as $\mathsf{cmt}_{1,\alpha^*} \xleftarrow{\$} \{0,1\}^{\ell(\lambda)}$. The probability of distinguishing Simulator 0 from Simulator 1 by any distinguisher running in time at most $\mathsf{t}(\lambda)$ is upper bounded by $\epsilon_{\mathsf{Com}}(\lambda)$.

Simulator 2. The only difference between Simulator 1 and Simulator 2 is that, Simulator 2 samples $\pi_{\alpha^*} \xleftarrow{\$} \mathcal{S}_n$ and $\mathbf{v}_{\alpha^*} \xleftarrow{\$} \mathbb{F}_q^n$ uniformly at random instead of using the seed-derived randomness from the seed θ. The probability of distinguishing Simulator 2 from Simulator 1 by any distinguisher running in time at most $\mathsf{t}(\lambda)$ is upper bounded by $\epsilon_{\mathsf{PRG}}(\lambda)$.

Simulator 3 (Sim). Simulator 3 takes the statement $x = (\mathbf{H}, (\mathbf{x}_i)_{i\in[1,t]}, (\mathbf{y}_i)_{i\in[1,t]})$, and the challenges (κ, α^*) as input and proceeds from Step 2 of Sim defined above. Note that, this simulator does not depend on the witness π.

If $\alpha^* = 1$, then Simulator 2 and Sim work exactly same till Step 5 of Sim. Therefore, h_1 is distributed identically in both the transcripts. Also, $\mathbf{s}_0$ is computed honestly by Sim and hence matches with that computed by Simulator 2. While computing $\mathbf{s}_1$, both Simulator 2 and Sim add $\mathbf{v}_1$ to it. However, $\mathbf{v}_1$ is sampled uniformly at random by both Simulator 2 and Sim. Hence, $\mathbf{s}_1$ is also distributed identically in both the transcripts. Step 11 of Sim works exactly as Simulator 2 which means h_2 is also distributed identically in both the transcripts. The response rsp in this case is $\mathsf{rsp} = \big(\mathbf{s}_1, (r_1 \,||\theta_i)_{i\in[2,N]}, \mathsf{cmt}_{1,1}\big)$. As explained above $\mathbf{s}_1$ is uniform random and distributed identically in both transcripts. The seeds $(\theta_i)_{i\in[2,N]}$ and randomness r_1 are computed identically by both the simulators since they work exactly the same way till Step 5 of Sim. Also, $\mathsf{cmt}_{1,1}$ is sampled uniformly at random in both experiments (refer Simulator 1). Therefore, the transcript $(x, h_1, (\kappa_i)_{i\in[1,t]}, h_2, \alpha^*, \mathsf{rsp})$ is distributed identically in Simulator 2 and Sim when $\alpha^* = 1$.

If $\alpha^* \neq 1$, then Simulator 2 and Sim work exactly same till Step 5 of Sim, except for sampling of π_1. Simulator 2 computes π_1 from witness π, whereas Sim samples π_1 uniformly at random. However, the values $r_1, \mathbf{v}, \mathsf{cmt}_1$, and h_1 are computed independently of π_1 and those are distributed identically in both the transcripts. Also note Simulator 2 computes π_1 by composing it with $\pi_{\alpha^*}^{-1}$. Since π_{α^*} is sampled uniformly at random by Simulator 2, this implies that π_1 computed by Simulator 2 is also uniform random permutation and hence π_1 is also distributed identically. Also, for $i \in [0, \alpha^* - 1]$, the values $\mathbf{s}_i$ are computed honestly by Sim and since π_1 is distributed identically the values $\mathbf{s}_i$ for $i \in [0, \alpha^* - 1]$ are also distributed identically. As in the previous case, while computing $\mathbf{s}_{\alpha^*}$, both Simulator 2 and Sim add $\mathbf{v}_{\alpha^*}$ to it. However, $\mathbf{v}_{\alpha^*}$ is sampled uniformly at random by both Simulator 2 and Sim. Hence, $\mathbf{s}_{\alpha^*}$ is also distributed identically in both the transcripts. Step 11 of Sim works exactly as Simulator 2 which means h_2 is also distributed identically in both the transcripts. The response rsp in this case is $\mathsf{rsp} = \big(\mathbf{s}_{\alpha^*}, (\pi_1 \,||r_1\,||\theta_i)_{i\in[1,N]\setminus\alpha^*}, \mathsf{cmt}_{1,\alpha^*}\big)$. As explained above $\mathbf{s}_{\alpha^*}$ is uniform random and distributed identically in both transcripts. The seeds

$(\theta_i)_{i\in[1,N]\setminus\alpha^*}$ and randomness r_1 are computed identically by both the simulators since they work exactly the same way till Step 5 of Sim. Also, $\mathsf{cmt}_{1,\alpha^*}$ is sampled uniformly at random in both experiments (refer Simulator 1). Therefore, the transcript $(x, h_1, (\kappa_i)_{i\in[1,t]}, h_2, \alpha^*, \mathsf{rsp})$ is distributed identically in Simulator 2 and Sim when $\alpha^* \neq 1$.

Therefore, any distinguisher running in time at most $\mathsf{t}(\lambda)$ cannot distinguish between the real-world transcript and the transcript produced by Sim with probability better than $(\epsilon_{\mathsf{PRG}}(\lambda) + \epsilon_{\mathsf{Com}}(\lambda))$.

□

6.2 Security Proof for the Signature Scheme

6.2.1 Proof of Theorem 3.4 We restate the Theorem 3.4 below and follow it by its proof.

Theorem 3.4. *Suppose* PRG *is* $(\mathsf{t}, \epsilon_{\mathsf{PRG}})$*-secure and any adversary running in time* $\mathsf{t}(\lambda)$ *can solve the the underlying* $\mathsf{r\text{-}IPKP}$ *instance with probability at most* $\epsilon_{\mathsf{r\text{-}IPKP}}$*. Model* H_0*,* H_1*, and* H_2 *as random oracles where* H_0*,* H_1*, and* H_2 *have* 2λ*-bit output length. Then chosen-message attacker against the signature scheme (PERK) presented in Figure 4, running in time* $\mathsf{t}(\lambda)$*, making* q_s *signing queries, and making* q_0*,* q_1*,* q_2 *queries, respectively, to the random oracles, succeeds in outputting a valid forgery with probability*

$$\Pr[\mathsf{Forge}] \leq \frac{(q_0 + \tau \cdot (N+1) \cdot q_s)^2}{2 \cdot 2^{2\lambda}} + \frac{q_s \cdot (q_0 + q_1 + q_2 + q_s)}{2^{2\lambda}} + \tau \cdot q_s \cdot \epsilon_{\mathsf{PRG}}(\lambda) + \epsilon_{\mathsf{r\text{-}IPKP}} + q_2 \cdot \varepsilon_{\mathsf{KS}}^{\tau}, \tag{5}$$

where $\varepsilon_{\mathsf{KS}} = \frac{1}{N} + \frac{N-1}{N\cdot(q^t-1)}$.

The following proof is greatly inspired from the proof of the Picnic signature scheme [CDG+20, Theorem 6.2] and [FJR22a, FJR22b, Theorem 5].

Proof (Proof of Theorem 3.4). Let $\mathcal{A}$ be a EUF-CMA attacker against the signature scheme, which makes q_s queries to the signing oracle. Also, let q_0, q_1, and q_2 respectively denote the number of queries made by $\mathcal{A}$ to the random oracles H_0, H_1, and H_2. To prove security we define a sequence of experiments involving $\mathcal{A}$, starting with an experiment in which $\mathcal{A}$ interacts with the real signature scheme. We let $\Pr_i[\cdot]$ refer to the probability of an event in experiment i. We let $\mathsf{t}(\lambda)$ denote the running time of the entire experiment, *i.e.*, including both $\mathcal{A}$'s running time and the time required to answer signing queries and to verify $\mathcal{A}$'s output.

Experiment 1. This corresponds to the interaction of $\mathcal{A}$ with the real signature scheme. In more detail: first KeyGen is run to obtain, the secret key $\mathsf{sk} = \pi$ along with the public key $\mathsf{pk} = (\mathbf{H}, (\mathbf{x}_j, \mathbf{y}_j)_{j\in[1,t]})$, and $\mathcal{A}$ is given pk. In addition, we assume that the random oracles H_0, H_1, and H_2 are chosen uniformly from the appropriate spaces. $\mathcal{A}$ may make signing queries, which will be answered as in

the signature algorithm; $\mathcal{A}$ may also query any of the random oracles. Finally, $\mathcal{A}$ outputs a message-signature pair; we let Forge denote the event that the message was not previously queried by $\mathcal{A}$ to its signing oracle, and the signature is valid. Our goal is to upper-bound $\Pr_1[\mathsf{Forge}]$.

Experiment 2. We abort the experiment if, during the course of the experiment, a collision occurs in H_0. The number of queries to any oracle throughout the experiment (by either the adversary or the signing algorithm) is at most $(q_0 + \tau \cdot (N+1) \cdot q_s)$. Therefore,

$$|\Pr_1[\mathsf{Forge}] - \Pr_2[\mathsf{Forge}]| \leq \frac{(q_0 + \tau \cdot (N+1) \cdot q_s)^2}{2 \cdot 2^{2\lambda}}.$$

Experiment 3. We abort the experiment if during the course of the experiment, while answering to a signature query, the sampled salt collides with the value salt in any previous query to H_0, H_1, or H_2. For each single signature query, the probability to abort is upper bounded by $(q_0 + q_1 + q_2 + q_s)/2^{2\lambda}$. Thus,

$$|\Pr_2[\mathsf{Forge}] - \Pr_3[\mathsf{Forge}]| \leq \frac{q_s \cdot (q_0 + q_1 + q_2 + q_s)}{2^{2\lambda}}.$$

Experiment 4. The difference with the previous experiment is that, when signing a message m we begin by choosing h_1 and h_2 uniformly at random and then we expand them as $(\kappa_j^{(e)})_{e\in[1,\tau],j\in[1,t]}$ and $(\alpha^{(e)})_{e\in[1,\tau]}$. Steps 1, 3, and 5 are computed as before, but in Steps 2 and 4 we simply set the output of H_1 to h_1 and the output of H_2 to h_2.

The outcome of this experiment compared to the previous one only changes if, in the course of answering a signing query, the query to H_1 or the query to H_2 was ever made before (by either the adversary or as a part of answering some other signing query). But this cannot happen since in such a case Experiment 3 would abort. Thus,

$$\Pr_3[\mathsf{Forge}] = \Pr_4[\mathsf{Forge}].$$

Experiment 5. The difference with the previous experiment is that, for each $e \in [1,\tau]$, we sample $\mathsf{cmt}^{(e)}_{1,\alpha^{(e)}}$ uniformly at random instead of making a query to H_0.

The only difference between this experiment and the previous experiment occurs if, during the course of answering a signing query, the seed $\theta^{(e)}_{\alpha^{(e)}}$ (for some $e \in [1,\tau]$) was previously queried to H_0. However, such collisions cannot occur within the same signing query (since indices e and i are part of the input of H_0) and if it occurs from a previous query (signing query or query to H_0) then the experiment aborts (according to the difference introduced in Experiment 3). Thus,

$$\Pr_4[\mathsf{Forge}] = \Pr_5[\mathsf{Forge}].$$

Experiment 6. We again modify the experiment. Now, for $e \in [1, \tau]$ the signer uses the SHVZK simulator Sim (see proof of Theorem 3.3) to generate the views of the parties during the execution of Step 1 and Step 3. We denote by $\mathsf{Sim}_{\mathsf{salt}}(\cdot)$ a call to this simulator which appends salt to the sampled seed θ as input to PRG. This simulation results in $\left\{\left(\theta_i^{(e)}, \pi_i^{(e)}\right)\right\}_{i \neq \alpha^{(e)}}$ and $\left(\mathbf{s}_j^{(e)}\right)_{j \in [1,N]}$. Thus the signing queries are now answered as shown in 6.

Step 0

1. Sample $h_1 \xleftarrow{\$} \{0,1\}^{2\lambda}$ uniformly at random.
2. Sample $(\kappa_j^{(e)})_{e \in [1,\tau], j \in [1,t]} \longleftarrow \mathsf{PRG}(h_1)$.
3. Sample $h_2 \xleftarrow{\$} \{0,1\}^{2\lambda}$ uniformly at random.
4. Sample $(\alpha^{(e)})_{e \in [1,\tau]} \longleftarrow \mathsf{PRG}(h_2)$.
5. Sample a uniform random salt $\mathsf{salt} \xleftarrow{\$} \{0,1\}^{2\lambda}$.

Steps 1 and 3 For each iteration $e \in [1, \tau]$

1. $\left\{\left(\theta_i^{(e)}, \pi_i^{(e)}\right)\right\}_{i \neq \alpha^{(e)}}, \left(\mathbf{s}_j^{(e)}\right)_{j \in [1,N]} \leftarrow \mathsf{Sim}_{\mathsf{salt}}((\kappa_j^{(e)})_{e \in [1,\tau], j \in [1,t]}, (\alpha^{(e)})_{e \in [1,\tau]})$
2. Choose commitment $\mathsf{cmt}^{(e)}_{1,\alpha^{(e)}} \xleftarrow{\$} \{0,1\}^{2\lambda}$ uniform randomly.
3. For $i \neq \alpha^{(e)}$:
   - ◇ If $i \neq 1$, set $\mathsf{cmt}^{(e)}_{1,i} = \mathsf{H}_0(\mathsf{salt}, e, i, \theta_i^{(e)})$.
   - ◇ If $i = 1$, set $\mathsf{cmt}^{(e)}_{1,1} = \mathsf{H}_0(\mathsf{salt}, e, 1, \pi_1, \theta_1^{(e)})$.
4. Compute $\mathbf{Hv}^{(e)}$ as $\mathbf{Hv}^{(e)} = (\mathbf{H}(\mathbf{s}_N^{(e)}) - \sum_{i \in [1,t]} \kappa_i^{(e)} \cdot \mathbf{y}_i)$.
5. Set $\mathsf{cmt}_1^{(e)} = \mathsf{H}_0(\mathsf{salt}, e, \mathbf{Hv}^{(e)})$.

Steps 2 and 4

1. Set $\mathsf{H}_1(\mathsf{salt}, m, \mathsf{pk}, (\mathsf{cmt}_1^{(e)}, \mathsf{cmt}^{(e)}_{1,i})_{e \in [1,\tau], i \in [1,N]})$ equal to h_1.
2. Set $\mathsf{H}_2(\mathsf{salt}, m, \mathsf{pk}, h_1, (\mathbf{s}_i^{(e)})_{e \in [1,\tau], i \in [1,N]})$ equal to h_2.

Step 5 Output signature σ built as

1. For each iteration $e \in [1, \tau]$:
   - ◇ Compute $\mathbf{z}_1^{(e)} = \mathbf{s}_\alpha^{(e)}$
   - ◇ If $\alpha^{(e)} \neq 1$, $z_2^{(e)} = \pi_1^{(e)} \,||\, (\theta_i^{(e)})_{i \in [1,N] \setminus \alpha^{(e)}}$
   - ◇ If $\alpha^{(e)} = 1$, $z_2^{(e)} = (\theta_i^{(e)})_{i \in [1,N] \setminus \alpha^{(e)}}$
   - ◇ Compute $\mathsf{rsp}^{(e)} = (\mathbf{z}_1^{(e)}, z_2^{(e)}, \mathsf{cmt}^{(e)}_{1,\alpha^{(e)}})$
2. Compute $\sigma = (\mathsf{salt}, h_1, h_2, (\mathsf{rsp}^{(e)})_{e \in [1,\tau]})$

Fig. 6: Experiment 6: Answer to a signature query for a message m.

Note that the secret π is no longer used for generating signatures. Recall that an adversary against Sim has distinguishing advantage $\epsilon_{\mathsf{PRG}}(\lambda)$ (corresponding to execution time $\mathsf{t}(\lambda)$), since the commitments are built outside of the simulator. Therefore,

$$|\Pr_5[\mathsf{Forge}] - \Pr_6[\mathsf{Forge}]| \leq \tau \cdot q_s \cdot \epsilon_{\mathsf{PRG}}(\lambda).$$

Experiment 7. At any point during the experiment, we say that the execution e^* of a query

$$h_2 = \mathsf{H}_2(\mathsf{salt}, m, \mathsf{pk}, h_1, (\mathbf{s}_i^{(e)})_{e\in[1,\tau],i\in[1,N]})$$

defines a *correct witness* if the following four conditions are fulfilled:

1. h_1 was output by a previous query

$$h_1 = \mathsf{H}_1(\mathsf{salt}, m, \mathsf{pk}, (\mathsf{cmt}_1^{(e)}, \mathsf{cmt}_{1,i}^{(e)})_{e\in[1,\tau],i\in[1,N]}),$$

2. each $\mathsf{cmt}_{1,i}^{(e^*)}$ in this H_1-query was output by a previous query

$$\mathsf{cmt}_{1,i}^{(e^*)} = \mathsf{H}_0(\mathsf{salt}, e^*, i, \theta_i^{(e^*)})$$

for $i \in [2, N]$, and

$$\mathsf{cmt}_{1,1}^{(e^*)} = \mathsf{H}_0(\mathsf{salt}, e^*, 1, \pi_1^{(e^*)}, \theta_1^{(e^*)})$$

for $i = 1$.

3. each $\mathsf{cmt}_1^{(e^*)}$ in the above H_1-query was output by a previous query

$$\mathsf{cmt}_1^{(e^*)} = \mathsf{H}_0\left(\mathsf{salt}, e^*, \left(\mathbf{Hs}_N^{(e^*)} - \sum_{i\in[1,t]} \kappa_i^{(e^*)} \cdot \mathbf{y}_i\right)\right)$$

4. the permutation π derived from $\left\{\pi_i^{(e^*)}\right\}_{i\in[1,N]}$ *i.e.* $\pi = \pi_N^{(e^*)} \circ \pi_{N-1}^{(e^*)} \circ \cdots \pi_1^{(e^*)}$ satisfies

$$\mathbf{H}\left(\pi\left[\sum_{i\in[1,t]} \kappa_i \cdot \mathbf{x}_i\right]\right) = \sum_{i\in[1,t]} \kappa_i \cdot \mathbf{y}_i$$

for some $\kappa \in {\mathbb{F}_q}^t \setminus \mathbf{0}$, where $\kappa := \{\kappa_1, \ldots, \kappa_t\}$ and $\mathbf{0} \in {\mathbb{F}_q}^t$ is the all zero vector.

(Note that in all cases the commitments in the relevant prior H_1-query, if it exists, must be unique since the experiment is aborted if there is ever a collision in H_0.)

In Experiment 7, for each query of the above form made by the adversary to H_2 (where m was not previously queried to the signing oracle), check if there exists an execution e^* which defines a correct witness. We let Solve be the event that this occurs for some query to H_2. Note that, if that event occurs, the $\left\{\pi_i^{(e^*)}\right\}_{i\in[1,N]}$ (which can be determined from the oracle queries of $\mathcal{A}$) allow the computation of solution to $\mathsf{r\text{-}IPKP}$. Therefore, $\Pr_7[\mathsf{Solve}] \leq \epsilon_{\mathsf{r\text{-}IPKP}}$. We claim that

$$\Pr_7\left[\mathsf{Forge} \bigwedge \overline{\mathsf{Solve}}\right] \leq q_2 \cdot \varepsilon_{\mathsf{KS}}^{\tau},$$

where $\varepsilon_{\mathsf{KS}} = \frac{1}{N} + \frac{N-1}{N\cdot(q^t-1)}$ is the knowledge soundness error of one execution. To see this, assume Solve does not occur. Then there is no execution of any H_2-query which defines a correct witness. When considering an arbitrary execution $e \in [1, \tau]$, the attacker can only possibly generate a forgery (using this H_2-query) if

1. $\mathcal{A}$ guesses the first challenge $\kappa^{(e^*)} \in {\mathbb{F}_q}^t \setminus \mathbf{0}$, where $\kappa^{(e^*)} := \left\{\kappa_1^{(e^*)}, \ldots, \kappa_t^{(e^*)}\right\}$ and $\mathbf{0} \in {\mathbb{F}_q}^t$ is the all zero vector.
2. or even if $\mathsf{cmt}_1^{(e^*)} \neq \mathsf{H}_0(\mathsf{salt}, e^*, (\mathbf{Hs}_N^{(e^*)} - \sum_{i\in[1,t]} \kappa_i^{(e^*)} \cdot \mathbf{y}_i))$ the attacker guesses the second challenge α^* such that the views of all remaining $N-1$ parties are consistent.

Thus, the overall probability with which the attacker can generate a forgery using this H_2-query is

$$\varepsilon_{\mathsf{KS}}^{\tau} = \left(\frac{1}{q^t - 1} + \left(1 - \frac{1}{q^t - 1}\right) \cdot \frac{1}{N}\right)^{\tau}.$$

The final bound is obtained by taking a union bound over all queries to H_2. This concludes the proof of Theorem 3.4. □

7 Known Attacks

7.1 Generic Attacks against Fiat-Shamir Signatures

Kales and Zaverucha in [KZ20], showed a generic attack on the non-interactive version of 5-round PoK schemes. The attack strategy is to guess either one of the challenges (ch_1 or ch_2) correctly which permits the prover to cheat. The attacker then aims to split the work by attempting to guess the first challenge for η^* instances out of τ parallel repetitions, and tries to guess ch_2 for the remaining $(\tau - \eta^*)$ instances.

If the attacker can guess η^* challenges for the first phase correctly, then he can answer all the N possible challenges for the $\alpha^{(e)}$ for those instances. Subsequently, to successfully cheat he has to guess the remaining $(\tau - \eta^*)$ values of $\alpha^{(e)}$ correctly.

The parameter η^* allows the attacker to balance the cost for both guessing phases. The overall cost is minimized for a choice of

$$\eta^* = \underset{0\leq\eta\leq\tau}{\arg\min} \left\{\frac{1}{P_1(\eta, \tau, q, t)} + N^{(\tau-\eta)}\right\} \tag{6}$$

where,

$$P_1(\eta, \tau, q, t) := \sum_{j=\eta}^{\tau} \left(\frac{1}{q^t - 1}\right)^j \left(\frac{q^t - 2}{q^t - 1}\right)^{\tau - j} \binom{\tau}{j}.$$

Finally, the total cost for the attacker is thus

$$\mathcal{W}_{KZ} = \frac{1}{P_1(\eta^*, \tau, q, t)} + N^{(\tau-\eta^*)}. \tag{7}$$

In [KZ20] the authors classify the 5-round protocols based on whether it is possible for the verifier to detect if the tuple $(\mathsf{cmt}, \mathsf{ch}_1, \mathsf{rsp}_1)$ is valid or not.

If the verifier can detect the validity of this tuple then the scheme is said to possess *early abort* property. The cost of the attack varies for different schemes based on whether they satisfy the early abort property or not. Our protocol and signature scheme do not possess the early abort property and hence the expected cost of attacking the PERK signature scheme proposed in this work, is given by Equation (7)

7.2 Attacks on the Permuted Kernel Problem

Despite its long standing history in cryptographic applications [Sha90, BFK+19, Beu20, BG23] and consequently many cryptanalytic efforts [Geo92, BCCG93, PC94, JJ01, LP11, KMP19, SBC22], algorithms to solve the Permuted Kernel Problem are still rather simple adaptations of combinatorial enumeration and meet-in-the-middle techniques. Even though there has been recent progress on attacks [SBC22], for the mono-dimensional case of IPKP, i.e., Definition 2.19 with $t = 1$, those attacks do not have a serious effect. For parameters matching NIST's security definition I, III and V they yield a reduced security level of roughly 2,1 and 0 bits. The authors of [SBC22] also study the more general multi-dimensional case of IPKP where $t > 1$ pairs $(\mathbf{x}_i, \mathbf{y}_i)$ are provided and a permutation π is searched that simultaneously satisfies $\mathbf{H}\pi(\mathbf{x}_i) = \mathbf{y}_i \forall i \in [t]$.

The difference between IPKP and the r-IPKP (on which our scheme is based) is that the solution in the case of r-IPKP does not necessarily have to be the permutation that works for all the given pairs, but it has to satisfy the PKP identity only for an arbitrary (non-zero) linear combination of those pairs, i.e. any permutation π such that

$$\mathbf{H}\pi\left(\sum_i \kappa_i \mathbf{x}_i\right) = \sum_i \kappa_i \mathbf{y}_i,$$

for a choice of the $\kappa_i \in \mathbb{F}_q$, is a solution. Clearly any algorithm applicable to IPKP can also be applied to find a solution to the r-IPKP problem. However, a solution to r-IPKP does not necessarily have to be a solution to IPKP. Therefore algorithms to solve r-IPKP split into those initially proposed for IPKP and those specifically designed to solve r-IPKP.

7.2.1 Attacks on IPKP The IPKP problem was introduced by Shamir in 1990 [Sha90]. Still, the best attack on mono-dimensional IPKP is a meet-in-the-middle adaptation known as the KMP algorithm by Koussa, Macario-Rat and Patarin [KMP19]. This algorithm extends easily to $t > 1$, which was recently formalized in [SBC22]. The multi-dimensional IPKP first appeared in the literature in 2011 [LP11]. However, until recently cryptanalysis only resulted in better algorithms for the particular case of binary fields [PT21]. Recently, Santini, Baldi and Chiraluce [SBC22] proposed the SBC algorithm which extends the KMP algorithm by a pre-processing step. For $t > 1$, i.e., for the multi-dimensional case, this results in improvements over the KMP approach. For the

concrete complexity estimation of those attacks we rely on the *CryptographicEstimators* library[2] [EVZB23], which incorporates (more efficient) versions of the scripts from the original publications. We give a brief overview of those attacks in Appendix A.1.

Relation between IPKP *and the Code Equivalence Problem.* In their recent work Santini et al. [SBC22] observed a link between the homogeneous variant of IPKP, i.e, when it holds $\mathbf{y}_i = \mathbf{0}$ for all i, and the (sub-)code equivalence problem. For the homogeneous variant, this leads to further restrictions on the parameters to guarantee the hardness of the problem. However, for the inhomogeneous variant, which we consider, this does not apply. We give details on this relation and why it does not translate to IPKP in Appendix A.3.

7.2.2 Attacks on r-IPKP We introduce the r-IPKP problem together with our scheme. However, it is still very related to the multi-dimensional and mono-dimensional versions of IPKP and their corresponding hardness. We already discussed the relation to the *multi-dimensional* case of IPKP.

Let us now focus on the relation between r-IPKP and the *mono-dimensional* version of IPKP. In this context r-IPKP can be seen as a multi-instance version of IPKP.

Therefore disregard the (most likely) unique solution to r-IPKP which simultaneously solves IPKP for the same t, i.e. the permutation that works for all pairs $(\mathbf{x}_i, \mathbf{y}_i)$. Further, assume that for any of the given pairs $(\mathbf{x}_i, \mathbf{y}_i)$ there exist a permutation π_i solving the corresponding mono-dimensional IPKP instance, that is a π_i that satisfies $\mathbf{H}\pi_i(\mathbf{x}_i) = \mathbf{y}_i$. If we would be forced to recover one of the π_i, this would exactly be a multi-instance version of IPKP. However, the r-IPKP allows to recover any permutation that works for an arbitrary linear combination of the given pairs. Clearly, this gives a total of q^t different pairs, but unlike the multi-instance case those pairs are related.

In fact, from a coding theory perspective the r-IPKP asks to recover a permutation that works for any codeword and the corresponding syndrome where the code is defined by the generator matrix containing the $\mathbf{x}_i$s as rows. We give a new algorithm that exploits this view on the r-IPKP in Appendix A.2. The core idea is to search for a low-weight codeword first and focus on solving the corresponding mono-dimensional IPKP instance specified by this codeword. The lower weight of the codeword leads to a reduced enumeration effort.

As one would expect, the complexity of this algorithm decreases with increasing t. However, for small choices of t, attacks that exploit the link to the multi-dimensional IPKP from the previous section have a lower complexity. Intuitively, in those cases r-IPKP is too close to a multi-instance IPKP as that the extra freedom could outweigh the gain from the extra restrictions on the solution given by the multi-dimensional case of IPKP.

[2] https://github.com/Crypto-TII/cryptographic_estimators

7.3 Concrete Complexity of solving r-IPKP

In this section we give details on how the variation of different parameters affects the hardness of r-IPKP. A solid understanding of those effects is crucial for secure parameter selection. As outlined previously, for solving the r-IPKP one can either directly apply an IPKP algorithm or solve one of the single IPKP instances defined by any linear combination of input pairs. Which of the two attack strategies is better depends on the particular choice of parameters.

Effect of the number of solutions. Multiple existing solutions can lead to a maximum speedup that is linear in this amount of solutions. Whether this maximum speedup can be realized depends on the particular algorithm. However, for our parameter selection we conservatively assume that any algorithm can leverage this maximum speedup.

Note that the expected number of solutions differs for the considered sub-problems. The expected number of solutions for any random IPKP instance is about $\mathtt{Sol}_{\mathsf{IPKP}} = \frac{n!}{q^{m \cdot t}}$. Note that the *mono*-dimensional IPKP instance solved in the context of our new Algorithm 1 is not random but contains z zeros. In this case the amount of expected solutions is only $\mathtt{Sol}^{\text{mono}}_{\mathsf{IPKP},z} = \frac{n!}{q^m z!}$.

Effect of t on the running time. Santini et al. [SBC22] observed that the running time of the KMP algorithm as well as the running time of their SBC algorithm for IPKP is asymptotically independent of t. For concrete parameters, these algorithms still yield speedups for increasing t but the running time quickly converges to a stable minimum. Therefore, based on known algorithms the hardness of IPKP does not seem to deteriorate for high values of t. Contrary, the complexity of our new Algorithm 1 is monotonically decreasing for increasing t. This shows that high choices of t in the r-IPKP context are vulnerable.

To visualize this we consider in Table 5 a fixed instance for increasing t. The SBC algorithm reaches its minimum running time already for $t = 10$, while Algorithm 1 constantly improves. However, the SBC algorithm has a lower complexity for small choices of t and obtains a larger gain for early increases. Note that for the chosen parameters in Table 5 already a random mono-dimensional IPKP instance has at most one solution in expectation, i.e. $\mathtt{Sol}^{\text{mono}}_{\mathsf{IPKP}} \leq 1$.

Effect of m on the running time. Generally the hardness of IPKP is increasing with decreasing m. This holds up to the point where there exist multiple solutions. Previous parameter selection for PKP-based schemes therefore chooses m minimal such that there exists no more than one random solution in expectation. However, for the specific case of the r-IPKP problem, the two sub-problems, i.e., the multi- and mono-dimensional IPKP instances, have a different amount of expected solutions. Here, decreasing m leads, generally, only to an increase of the problem complexity as long as the solution to both sub-problems is still unique.

t	T_{new}	T_{SBC}
1	139.35	139.35
2	139.01	116.25
3	137.46	110.70
10	115.98	88.18
15	100.27	88.18
19	89.14	88.18
20	85.63	88.18
25	71.77	88.18
30	63.47	88.18

Table 5: Bit complexity of SBC algorithm (T_{SBC}) and Algorithm 1 (T_{new}) for increasing t on r-IPKP(n, m, q, t) instance with $(n, m, q) = (66, 31, 1021)$.

Parameter selection. For parameter selection we ensure that the complexity of the SBC algorithm as well as the complexity of our new Algorithm 1 are above the security threshold, when assuming a linear speedup from the existing amount of solutions. Note that it is important to consider both strategies as the IPKP suggests to decrease m to increase the difficulty of the problem. This is related to the low amount of expected solutions $\texttt{Sol}_{\mathsf{IPKP}}$, which allows to decrease m significantly without introducing multiple solutions. Contrary, for any mono-dimensional instance given by the possible linear combinations, there exist several solutions $\texttt{Sol}^{\text{mono}}_{\mathsf{IPKP}}$ for such small choices of m, which decrease the complexity of Algorithm 1.

Note that we, conservatively, restrict in our parameter selection to small choices of $t \in \{3, 5\}$ to guard against attacks that exploit the specifics of r-IPKP over IPKP. For such small values of t, the SBC algorithm has generally a lower complexity than Algorithm 1 (compare to Table 5). In those cases, the parameter selection process leads to a choice of m which implies a unique solution to the multi-dimensional IPKP instance, while there exist multiple solutions to the mono-dimensional instance solved in the context of Algorithm 1. This leads to a balancing of both complexities via the amount of solutions.

Our complexity estimations ignore polynomial factors and ensure that already the exponential factors of the complexity formulas reach the NIST security levels of 143, 207 and 272 bits of category I, III and V respectively. For the complexity estimation of the SBC algorithm we rely on the *CryptographicEstimators* library[3] incorporating a more efficient version of the script from [SBC22]. For our algorithm we rely on a separate script. Overall this leads to the choices of parameters given in Section 4. We restate those parameter sets in Table 6 together with the complexity estimation following the mentioned methodology.

Note that even though we are quite conservative in parameter selection by restricting to small choices of t, disregarding polynomial factors and assuming

[3] https://github.com/Crypto-TII/cryptographic_estimators

Parameter Set	q	n	m	t	T_{new}	T_{SBC}
PERK-I-3	1021	79	35	3	147	146
PERK-I-5	1021	83	36	5	147	148
PERK-III-3	1021	112	54	3	210	211
PERK-III-5	1021	116	55	5	211	213
PERK-V-3	1021	146	75	3	275	275
PERK-V-5	1021	150	76	5	276	274

Table 6: Bit complexity estimation for different PERK parameter sets.

a maximum speedup from multiple solutions, we obtain competitive signature sizes. Also all considered algorithms use as much memory as they consume time, which in a more realistic estimate that accounts for memory access leads to an even higher security margin.

8 Advantages and Limitations

We now discuss some advantages and limitations of PERK.

8.1 Advantages

Some advantages of our design are:

+ PERK features very small public key and secret key sizes along with moderate signature sizes. Therefore, on the combined metric of pk + signature size, PERK produces sizes of approximately 6kB for NIST Security Level I which compares well with other signature schemes.

+ PERK performances are constrained by numerous calls to symmetric cryptographic primitives. Any speedup to the implementation of these primitives directly benefit PERK. In particular, hardware acceleration support for such primitives improves the performance of the scheme.

+ Many efficient post-quantum schemes rely on difficult problems featuring additional structure, such as a (quasi-)cyclic or ring structure. This comes at the cost of less-tight security reductions and potential new attack vectors. In contrast PERK does not introduce objects with such special structure.

+ Resilience against PKP and r-IPKP attacks: A large part of the signature size scales with the security parameter λ (due to the seed trees and commitments) and not directly with the r-IPKP parameters. As a consequence, increasing the r-IPKP parameters has a limited impact on the total size of the signature.

8.2 Limitations

In the following, we point out the limitations of PERK.

- While PERK's performance profile is comparable to other MPCitH-based constructions, those can not compete with the fastest post-quantum secure schemes, usually based on structured lattices.
- The security of PERK is based on a new variant of the PKP (r-IPKP) introduced together with the scheme. However, we note that the r-IPKP only slightly deviates from the case of PKP especially in the case of smaller values of t, such as $t = 3$.
- Since PERK follows the MPC-in-the-head paradigm to construct the zero-knowledge proof of knowledge system, it inherits the generic limitations of such designs. Namely, even if the objects related to the computationally hard problem could be compressed arbitrarily, it would not lead to arbitrarily short signatures, since the amount of auxiliary information is already a significant fraction of the total signature size.

References

ACK21. Thomas Attema, Ronald Cramer, and Lisa Kohl. A compressed Σ-protocol theory for lattices. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part II*, volume 12826 of *LNCS*, pages 549–579, Virtual Event, August 2021. Springer, Heidelberg.

AF22. Thomas Attema and Serge Fehr. Parallel repetition of $(k_1, \ldots, k_\mu)$-special-sound multi-round interactive proofs. In Yevgeniy Dodis and Thomas Shrimpton, editors, *CRYPTO 2022, Part I*, volume 13507 of *LNCS*, pages 415–443. Springer, Heidelberg, August 2022.

AFK22. Thomas Attema, Serge Fehr, and Michael Klooß. Fiat-shamir transformation of multi-round interactive proofs. In Eike Kiltz and Vinod Vaikuntanathan, editors, *TCC 2022, Part I*, volume 13747 of *LNCS*, pages 113–142. Springer, Heidelberg, November 2022.

BCCG93. Thierry Baritaud, Mireille Campana, Pascal Chauvaud, and Henri Gilbert. On the security of the permuted kernel identification scheme. In Ernest F. Brickell, editor, *CRYPTO'92*, volume 740 of *LNCS*, pages 305–311. Springer, Heidelberg, August 1993.

BDG20. Mihir Bellare, Hannah Davis, and Felix Günther. Separate your domains: NIST PQC KEMs, oracle cloning and read-only indifferentiability. In Anne Canteaut and Yuval Ishai, editors, *EUROCRYPT 2020, Part II*, volume 12106 of *LNCS*, pages 3–32. Springer, Heidelberg, May 2020.

Ber19. Daniel J. Bernstein. djbsort. https://sorting.cr.yp.to/, 2019. [Online; accessed 20-June-2023].

Beu20. Ward Beullens. Sigma protocols for MQ, PKP and SIS, and Fishy signature schemes. In Anne Canteaut and Yuval Ishai, editors, *EUROCRYPT 2020, Part III*, volume 12107 of *LNCS*, pages 183–211. Springer, Heidelberg, May 2020.

BFK+19. Ward Beullens, Jean-Charles Faugère, Eliane Koussa, Gilles Macario-Rat, Jacques Patarin, and Ludovic Perret. PKP-based signature scheme. In Feng Hao, Sushmita Ruj, and Sourav Sen Gupta, editors, *INDOCRYPT 2019*, volume 11898 of *LNCS*, pages 3–22. Springer, Heidelberg, December 2019.

BG23. Loïc Bidoux and Philippe Gaborit. Compact post-quantum signatures from proofs of knowledge leveraging structure for the PKP, SD and RSD problems. In *Codes, Cryptology and Information Security (C2SI)*, pages 10–42. Springer, 2023.

BGK17. Thierry P. Berger, Cheikh Thiécoumba Gueye, and Jean Belo Klamti. A np-complete problem in coding theory with application to code based cryptography. In Said El Hajji, Abderrahmane Nitaj, and El Mamoun Souidi, editors, *Codes, Cryptology and Information Security - Second International Conference, C2SI 2017, Rabat, Morocco, April 10-12, 2017, Proceedings - In Honor of Claude Carlet*, volume 10194 of *Lecture Notes in Computer Science*, pages 230–237. Springer, 2017.

CDG+20. Melissa Chase, David Derler, Steven Goldfeder, Daniel Kales, Jonathan Katz, Vladimir Kolesnikov, Claudio Orlandi, Sebastian Ramacher, Christian Rechberger, Daniel Slamanig, Xiao Wang, and Greg Zaverucha. The picnic signature scheme design document (version 3.0). Available at https://csrc.nist.gov/CSRC/media/Projects/post-quantum-cryptography/documents/round-3/submissions/Picnic-Round3.zip (Last checked on: May 24, 2023), 2020.

EVZB23. Andre Esser, Javier Verbel, Floyd Zweydinger, and Emanuele Bellini. `CryptographicEstimators`: a software library for cryptographic hardness estimation. *Cryptology ePrint Archive*, 2023.

FJR22a. Thibauld Feneuil, Antoine Joux, and Matthieu Rivain. Syndrome decoding in the head: Shorter signatures from zero-knowledge proofs. In Yevgeniy Dodis and Thomas Shrimpton, editors, *CRYPTO 2022, Part II*, volume 13508 of *LNCS*, pages 541–572. Springer, Heidelberg, August 2022.

FJR22b. Thibauld Feneuil, Antoine Joux, and Matthieu Rivain. Syndrome decoding in the head: Shorter signatures from zero-knowledge proofs. Cryptology ePrint Archive, Report 2022/188, 2022. https://eprint.iacr.org/2022/188.

FJR23. Thibauld Feneuil, Antoine Joux, and Matthieu Rivain. Shared permutation for syndrome decoding: new zero-knowledge protocol and code-based signature. *Designs, Codes and Cryptography*, 91(2):563–608, 2023.

FS87. Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Andrew M. Odlyzko, editor, *CRYPTO'86*, volume 263 of *LNCS*, pages 186–194. Springer, Heidelberg, August 1987.

Geo92. Jean Georgiades. Some remarks on the security of the identification scheme based on permuted kernels. *Journal of Cryptology*, 5(2):133–137, January 1992.

IKOS07. Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai. Zero-knowledge from secure multiparty computation. In David S. Johnson and Uriel Feige, editors, *39th ACM STOC*, pages 21–30. ACM Press, June 2007.

IKOS09. Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai. Zero-knowledge proofs from secure multiparty computation. *SIAM Journal on Computing*, 39(3):1121–1152, 2009.

JJ01. Éliane Jaulmes and Antoine Joux. Cryptanalysis of PKP: A new approach. In Kwangjo Kim, editor, *PKC 2001*, volume 1992 of *LNCS*, pages 165–172. Springer, Heidelberg, February 2001.

KKW18. Jonathan Katz, Vladimir Kolesnikov, and Xiao Wang. Improved non-interactive zero knowledge with applications to post-quantum signatures. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *ACM CCS 2018*, pages 525–537. ACM Press, October 2018.

KMP19. Eliane Koussa, Gilles Macario-Rat, and Jacques Patarin. On the complexity of the permuted kernel problem. Cryptology ePrint Archive, Report 2019/412, 2019. https://eprint.iacr.org/2019/412.

KZ20. Daniel Kales and Greg Zaverucha. An attack on some signature schemes constructed from five-pass identification schemes. In Stephan Krenn, Haya Shulman, and Serge Vaudenay, editors, *CANS 20*, volume 12579 of *LNCS*, pages 3–22. Springer, Heidelberg, December 2020.

LP11. Rodolphe Lampe and Jacques Patarin. Analysis of some natural variants of the pkp algorithm. *Cryptology ePrint Archive*, 2011.

PC94. Jacques Patarin and Pascal Chauvaud. Improved algorithms for the permuted kernel problem. In Douglas R. Stinson, editor, *CRYPTO'93*, volume 773 of *LNCS*, pages 391–402. Springer, Heidelberg, August 1994.

PR97. Erez Petrank and Ron M. Roth. Is code equivalence easy to decide? *IEEE Trans. Inf. Theory*, 43(5):1602–1604, 1997.

Pra62. Eugene Prange. The use of information sets in decoding cyclic codes. *IRE Transactions on Information Theory*, 8(5):5–9, 1962.

PT21. Thales Bandiera Paiva and Routo Terada. Cryptanalysis of the binary permuted kernel problem. In Kazue Sako and Nils Ole Tippenhauer, editors, *ACNS 21, Part II*, volume 12727 of *LNCS*, pages 396–423. Springer, Heidelberg, June 2021.

SBC22. Paolo Santini, Marco Baldi, and Franco Chiaraluce. Computational hardness of the permuted kernel and subcode equivalence problems. Cryptology ePrint Archive, Report 2022/1749, 2022. https://eprint.iacr.org/2022/1749.

Sen00. Nicolas Sendrier. Finding the permutation between equivalent linear codes: The support splitting algorithm. *IEEE Transactions on Information Theory*, 46(4):1193–1203, 2000.

Sha90. Adi Shamir. An efficient identification scheme based on permuted kernels (extended abstract) (rump session). In Gilles Brassard, editor, *CRYPTO'89*, volume 435 of *LNCS*, pages 606–609. Springer, Heidelberg, August 1990.

A Details on Security

In this section we provide more details on the considered algorithms for parameter selection. We give a brief sketch of known algorithms to solve IPKP. After that we show how to adapt some of those techniques to the specific case of r-IPKP exploiting the extra degree of freedom. This results in improvements for certain regimes of parameters, where either many pairs are given or many solutions for single linear combinations exist. Afterwards we give details on the link between IPKP and the code equivalence problem.

A.1 Attacks on IPKP

In this section we briefly sketch the different approaches to solve the IPKP. Fully fledged descriptions, analysis and estimation scripts are given for example in [KMP19, SBC22, EVZB23].

The KMP Algorithm. The algorithm by Koussa, Macario-Rat and Patarin [KMP19] is a slight variant of previously known combinatorial techniques [Geo92, BCCG93, PC94, JJ01]. The algorithm was first proposed for the mono-dimensional IPKP [KMP19] and recently extended to the multi-dimensional case [SBC22]. We outline here the variant for mono-dimensional IPKP first.

Initially, the matrix $\mathbf{H}$ is transformed into semi-systematic form by applying a change of basis (modelled by the invertible matrix $\mathbf{Q}$)

$$\mathbf{QH} = \begin{pmatrix} \mathbf{I}_{m-u} & \mathbf{H}_1 \\ \mathbf{0} & \mathbf{H}_2 \end{pmatrix},$$

where $\mathbf{H}_1 \in \mathbb{F}_q^{(m-u)\times(n-m+u)}$,$\mathbf{H}_2 \in \mathbb{F}_q^{u\times(n-m+u)}$ and u is an optimization parameter of the algorithm. By multiplying the syndrome $\mathbf{y}$ by the same matrix $\mathbf{Q}$ one maintains the validity of the PKP identity

$$\begin{aligned}\mathbf{QH}\pi(\mathbf{x}) &= \begin{pmatrix} \mathbf{I}_{m-u} & \mathbf{H}_1 \\ \mathbf{0} & \mathbf{H}_2 \end{pmatrix} \pi(\mathbf{x}) = \begin{pmatrix} \mathbf{I}_{m-u} & \mathbf{H}_1 \\ \mathbf{0} & \mathbf{H}_2 \end{pmatrix} \begin{pmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \end{pmatrix} = (\mathbf{x}_1 + \mathbf{H}_1\mathbf{x}_2, \mathbf{H}_2\mathbf{x}_2)^\top \\ &= (\mathbf{y}_1, \mathbf{y}_2)^\top = \mathbf{Qy},\end{aligned}$$

where $\mathbf{Qy} = (\mathbf{y}_1, \mathbf{y}_2) \in \mathbb{F}_q^{m-u} \times \mathbb{F}_q^u$ and $\pi(\mathbf{x}) = (\mathbf{x}_1, \mathbf{x}_2) \in \mathbb{F}_q^{m-u} \times \mathbb{F}_q^u$. The algorithm now focuses on solving the identity $\mathbf{H}_2\mathbf{x}_2 = \mathbf{y}_2$. For any found $\mathbf{x}_2$ satisfying the identity it is than checked if $\mathbf{x}_1 = \mathbf{y}_1 - \mathbf{H}_1\mathbf{x}_2$ and $\mathbf{x}_2$ together form a permutation of $\mathbf{x}$.

Candidates for $\mathbf{x}_2$ are obtained by a meet-in-the-middle enumeration strategy. Therefore $\mathbf{x}_2$ is further split as $\mathbf{x}_2 = (\mathbf{x}_{21}, \mathbf{x}_{22})$, with $\mathbf{x}_{21}, \mathbf{x}_{22} \in \mathbb{F}_q^{u\times((n-m+u)/2)}$ to obtain the meet-in-the-middle identity

$$\mathbf{H}_2(\mathbf{x}_{21}, \mathbf{0}) = \mathbf{y}_2 - (\mathbf{0}, \mathbf{x}_{22}). \tag{8}$$

Then the algorithm enumerates all candidates for $\mathbf{x}_{21}$ and $\mathbf{x}_{22}$, that is all permutations of any selection of $(n-m+u)/2$ entries of $\mathbf{x}$. For each such vector the left (resp. right) side of Equation (8) is stored in a list L_1 (resp. L_2). In a final step the algorithm searches for matches between the lists L_1 and L_2 yielding the candidates for $\mathbf{x}_2$. From there $\mathbf{x}_1$ can be computed as $\mathbf{x}_1 = \mathbf{y}_1 - \mathbf{H}_1\mathbf{x}_2$. If $(\mathbf{x}_1, \mathbf{x}_2)$ forms a permutation of $\mathbf{x}$ this yields the solution π.

The complexity of the algorithm is (up to polynomial factors) linear in the sizes of the lists L_1, L_2 and L, where L is the list of matches. The expected sizes are

$$|L_1| = |L_2| = \binom{n}{(n-m+u)/2}\big((n-m+u)/2\big)! \quad \text{and} \quad |L| = \frac{|L_1 \times L_2|}{q^u}$$

Extension to multi-dimensional IPKP. The algorithm can easily be extended to solve IPKP for arbitrary t, as it was recently made explicit in [SBC22]. Therefore let $\mathbf{X}$ be the matrix containing the $\mathbf{x}_i$ as rows and $\mathbf{Y}$ containing the $\mathbf{y}_i$ as columns. Substituting the occurrences of $\mathbf{x}$ and $\mathbf{y}$ by $\mathbf{X}$ and $\mathbf{Y}$ resp., where the permutation now operates as a column permutation on matrices, one obtains this generalization. Then of course the definition of $(\mathbf{x}_1, \mathbf{x}_2)$ and $(\mathbf{y}_1, \mathbf{y}_2)$ analogously extends to matrices.

In terms of complexity, the enumeration effort stays (up to polynomial factors) exactly the same, as the possible number of permutations remains unchanged. The only difference is that the expected size of the list L of matches reduces to $\frac{|L_1 \times L_2|}{q^{u \cdot t}}$.

The SBC algorithm. The algorithmic improvement by Santini-Baldi-Chiraluce (SBC) extends the KMP algorithm by a preprocessing step.

Assume that the matrix $\mathbf{H}_2$ constructed in the KMP algorithm would contain zero columns. Clearly, those columns could be removed without affecting the validity of the identity $\mathbf{H}_2\mathbf{x}_2 = \mathbf{y}_2$. But in turn this would reduce the enumeration effort to find candidates for $\mathbf{x}_2$.

The preprocessing step of the SBC algorithm now consists in finding a u-dimensional subcode of $\mathbf{H}$ that has small support $w < n - m + u$, i.e., an $\mathbf{H}_2$ that contains some zero columns. This can be accomplished by adaptations of Information Set Decoding (ISD) algorithms. Subsequently, the SBC algorithm continues as the KMP algorithm by finding candidates for $\mathbf{x}_2$ in $\mathbf{H}_2\mathbf{x}_2 = \mathbf{y}_2$, now with reduced enumeration complexity. This resulting list of candidates is now treated as the first list, L_1, in the KMP algorithm.

Note that the KMP algorithm creates two lists each giving candidates for $(n - m + u)/2$ entries of the permutation. Now, as there are already candidates for w entries in L_1, the second list enumerates the permutation for further $n - m + u - w$ positions. Eventually both lists are matched on $u \cdot t$ coordinates as in the KMP algorithm to obtain a list of final candidates. Note that as in the KMP algorithm now each candidate of the final list reveals $n-m+u$ potential positions of the permutation which can be checked in polynomial time for extending to a full solution.

A.2 A new Algorithm solving r-IPKP

In the following we give a new algorithm for solving r-IPKP. This algorithm focuses on finding a permutation that solves the IPKP defined by any one linear combination of the input pairs, rather than finding the solution to the multi-dimensional IPKP instance. The algorithm is based on a preprocessing of the given pairs $(\mathbf{x}_i, \mathbf{y}_i)$, a subsequent instance permutation and an adapted KMP-style enumeration technique. Moreover, our algorithm contains the KMP algorithm as a special case.

The algorithm starts by finding a low Hamming-weight codeword in the code whose generator matrix contains the $\mathbf{x}_i$ as rows. This task is accomplished by

application of an ISD algorithm. Let $\mathbf{x}' = \sum \kappa_i \mathbf{x}_i$ be this codeword of weight w and $\mathbf{y}' = \sum \kappa_i \mathbf{y}_i$ the corresponding syndrome.

We now focus on finding a permutation π that satisfies $\mathbf{H}\pi(\mathbf{x}') = \mathbf{y}'$. Therefore, we apply a KMP-style enumeration with some modifications. Again we derive the identity $\mathbf{H}_2\mathbf{x}_2 = \mathbf{y}_2 \in \mathbb{F}_q^u$, with $\mathbf{x}_2 = (\mathbf{x}_{21}, \mathbf{x}_{22})$ as in the usual KMP algorithm. Now, recall that $\pi(\mathbf{x}') = (\mathbf{x}_1, \mathbf{x}_2)$ contains $n - w$ zeros. For the enumeration of $\mathbf{x}_{21}$ and $\mathbf{x}_{22}$ we now assume that z of those zeros are mapped into $\mathbf{x}_2$ by the permutation. Moreover, we assume that $z/2$ of those zeros are mapped to $\mathbf{x}_{21}$ and $z/2$ to $\mathbf{x}_{22}$. This leads to a reduced amount of candidates for $\mathbf{x}_{21}, \mathbf{x}_{22}$ that has to be enumerated.

Of course we do not know a priori if the permutation indeed distributes $z/2$ zeros onto $\mathbf{x}_{21}$ and $z/2$ zeros onto $\mathbf{x}_{22}$. Therefore prior to the enumeration we apply a random permutation to the columns of $\mathbf{H}$ to redistribute the weight (and zeros) of $\pi(\mathbf{x}')$. If the enumeration does not lead to a solution we repeat with a different column permutation of $\mathbf{H}$.

A pseudocode description is given in Algorithm 1. Note that for $w = n$, i.e., a maximum-weight codeword, we resemble the standard KMP algorithm for solving mono-dimensional IPKP.

Analysis of Algorithm 1. Let us start with the correctness of the algorithm.
Correctness. Note that the permuted instance $(\mathbf{H}^*, \mathbf{x}', \mathbf{y}')$ with $\mathbf{H}^* = \pi'(\mathbf{H})$ has solution $\pi' \circ \pi$ if π solves the original instance $(\mathbf{H}, \mathbf{x}', \mathbf{y}')$. Therefore the algorithm correctly returns $(\pi')^{-1} \circ \tilde{\pi}$ as the solution to the original instance, where $\tilde{\pi}$ solves the permuted instance.

Accordingly the solution to the permuted instance is $(\mathbf{x}_1, \mathbf{x}_2) = \pi'(\pi(\mathbf{x}'))$. Line 9 to 11 enumerate all candidates for $\mathbf{x}_2$ satisfying $\mathbf{H}_2\mathbf{x}_2 = \mathbf{y}_2$, where $\mathbf{x}_2 = (\mathbf{z}_1, \mathbf{z}_2)$ with each of the $\mathbf{z}_i$ containing $z/2$ zeros. Note that by construction $(\mathbf{x}_1, \mathbf{x}_2)$ contains $n - w$ zeros. As the permutation π' redistributes the zeros the algorithm can recover the permutation.
Complexity. The complexity of the algorithm splits into the cost of finding the short codeword $\mathbf{x}'$ and the cost of the `repeat` loop. The codeword is found by application of an ISD algorithm. Let us denote this cost by T_{ISD}.

The cost of the loop is equal to the amount of repetitions times the cost of one iteration. The amount of different permutations until the zeros are distributed as desired is

$$P = \frac{\binom{n}{n-w}}{\binom{n-2k}{n-w-z}\binom{k}{z/2}^2},$$

where $k = (n-m+u)/2$. The cost for one iteration is (up to polynomial factors) linear in the involved lists' sizes. Note that we have

$$|L_i| = \binom{k}{z/2}\binom{n-z}{k-z/2}(k-z/2)! \quad \text{and} \quad |L| = \frac{|L_1 \times L_2|}{q^u}.$$

The total time complexity therefore amounts to

$$T = \tilde{\mathcal{O}}\left(T_{\mathrm{ISD}} + (|L_1| + |L|) \cdot P\right),$$

Algorithm 1: Algorithm Solving r-IPKP

Input : r-IPKP instance $(\mathbf{H}, (\mathbf{x}_i, \mathbf{y}_i)_{i\in[t]})$

Output: solution π, $\kappa_i \in \mathbb{F}_q$, $i \in [t]$

For a vector $\mathbf{x}$ and an integer k let $\mathcal{P}_{\mathbf{x},k}$ be the set of vectors of length k with entries from $\mathbf{x}$ (with their maximum occurence as in $\mathbf{x}$).

Choose optimal positive $u \leq n-m$, $w \leq n$, and $z \leq n-w$, let $k := (n-m+u)/2$

Find weight-w codeword $\mathbf{x}' = \sum_i \kappa_i \mathbf{x}_i$ in the code defined by the $\mathbf{x}_i$

Let $\mathbf{y}' = \sum_i \kappa_i \mathbf{y}_i$

repeat

  choose random permutation π'

  $\mathbf{H}^* = \pi'(\mathbf{H})$

  $\mathbf{H}' = \mathbf{Q}\mathbf{H}^* = \begin{pmatrix} \mathbf{I}_{m-u} & \mathbf{H}_1 \\ \mathbf{0} & \mathbf{H}_2 \end{pmatrix}$, $(\mathbf{y}_1, \mathbf{y}_2) = \mathbf{Q}\mathbf{y}$

  $L_1 = \{\big(H_2(\mathbf{z}_1, 0^k), \mathbf{z}_1\big) \mid \mathbf{z}_1 \in \mathcal{P}_{\mathbf{x},k} \wedge \mathrm{wt}(\mathbf{z}_1) = k - z/2\}$

  $L_2 = \{\big(\mathbf{y}_2 - H_2(0^k, \mathbf{z}_2), \mathbf{z}_2\big) \mid \mathbf{z}_2 \in \mathcal{P}_{\mathbf{x},k} \wedge \mathrm{wt}(\mathbf{z}_2) = k - z/2\}$

  Compute $L = \{(\mathbf{z}_1, \mathbf{z}_2) \in L_1 \times L_2 \mid \mathbf{H}_2(\mathbf{z}_1, \mathbf{z}_2) = \mathbf{y}_2\}$ from L_1, L_2

  foreach $\mathbf{x}_2 \in L$ **do**

    $\mathbf{x}_1 = \mathbf{y}_1 - \mathbf{H}_1\mathbf{x}_2$

    if $\exists \tilde{\pi}\colon \tilde{\pi}(\mathbf{x}_1, \mathbf{x}_2) = \mathbf{x}'$ **then**

      return $(\pi')^{-1} \circ \tilde{\pi}$

while the memory complexity is equal to $M = \tilde{\mathcal{O}}\left(|L_1| + |L|\right)$.

In our numerical optimization we use for T_{ISD} the basic ISD procedure by Prange [Pra62] which gives

$$T_{\mathrm{ISD}} = \tilde{\mathcal{O}}\left(\frac{\binom{n}{w}}{\binom{n-m}{w}}\right).$$

There are more sophisticated ISD procedures with lower cost, but as T_{ISD} does not dominate the running time, we refrain from further optimizations.

Note that this running time assumes a single existing solution for the considered mono-dimensional instance solved. In case of multiple solutions the running time can be lower, which we discuss in Section 7.3.

A.3 Relation between IPKP and the Code Equivalence Problem

In their recent work Santini et al. [SBC22] also observed the relation between the the homogeneous variant of IPKP, i.e, when it holds $\mathbf{y}_i = \mathbf{0}$ for all i, and

the (sub-)code equivalence problem. For the (sub-) code equivalence problem one is given two generator matrices – one generator matrix $\mathbf{G}$ of an $[n, k]$ linear code and another $\mathbf{G}'$ of an $[n, k']$ code with $k' \leq k$. The problem now asks to find a subcode $\tilde{\mathbf{G}}$ of $\mathbf{G}$ of dimension k' that maps into $\mathbf{G}'$ via a permutation. Precisely, one is asked to find a subcode $\tilde{\mathbf{G}}$ of $\mathbf{G}$, an invertible matrix $\mathbf{S} \in \mathbb{F}_q^{k' \times k'}$, corresponding to a change of basis, and a permutation π such that $\mathbf{G}' = \mathbf{S}\pi(\tilde{\mathbf{G}})$. For $k' = k$ the problem is known as the *code equivalence problem*.

The homogeneous version of the IPKP is the dual formulation of the subcode equivalence problem that asks to first recover the permutation π. Therefore one might see the given matrix $\mathbf{H}$ as the parity check matrix of $\mathbf{G}$ and the matrix with the $\mathbf{x}_i$ as rows as the generator matrix $\mathbf{G}'$. The permuted kernel problem now asks to find a permutation of $\mathbf{G}'$, such that $\mathbf{H}\big(\pi(\mathbf{G}')\big)^\top = \mathbf{0}$, or in other words a permutation such that $\pi(\mathbf{G}')$ is contained in the code with parity-check matrix $\mathbf{H}$ as a subcode.

While the subcode equivalence problem, analogously to PKP is known to be NP-complete [BGK17], the special case of code-equivalence, i.e., $k = k'$ or in PKP notation $n - m = t$, is likely to be not [PR97]. In particular, there is a polynomial time algorithm known to solve random instances of the code equivalence problem [Sen00], that is instances where $\mathbf{G}$ and $\mathbf{G}'$ are chosen at random. Therefore, if one wants to ensure difficult instances, one needs to ensure $n - m \neq t$. More specifically, one also needs to make sure that the difference $(n - m) - t$ is large enough, as there is a reduction from subcode equivalence to code equivalence with an overhead of $q^{t(n-m-t)}$ [SBC22].

Note that this whole equivalence relation only holds for the *homogeneous* variant of IPKP and not for the *inhomogeneous* variant. Therefore to make use of the mentioned equivalence in our case of IPKP one has to first apply a reduction from the IPKP to its homogeneous variant. This reduction exists, but decreases m by t. Essentially, one has to add those codewords $\mathbf{c}_i$ that yield the corresponding syndromes, i.e., those with $\mathbf{H}\mathbf{c}_i = \mathbf{y}_i$, to the code with parity-check matrix $\mathbf{H}$. The parity-check matrix $\mathbf{H}'$ of this enlarged code together with the pairs $(\mathbf{x}_i, \mathbf{y}_i$ then define a homogeneous IPKP instance. Note that $\mathbf{H}'$ now has $m' = m - t$ rows. In turn the relation $n - m \neq t$ from the homogeneous setting becomes trivial in the inhomogeneous setting, namely

$$n - m' \neq t \Leftrightarrow n - m \neq 0.$$

This implies that the inhomogeneous case is inherently harder than the homogeneous case.