Post-Quantum Private Set Intersection for Small Sets

Junxin Liu Mike Rosulek Ni Trieu

September 11, 2026

Abstract

Are private set intersection (PSI) protocols ready for the post-quantum future? We focus on the PSI protocol of Rosulek & Trieu ("RT21", ACM CCS 2021), which is the current state-of-the-art for PSI on small sets (less than a thousand items). The RT21 protocol presents some fundamental barriers to post-quantum security. First, although it is written in terms of an arbitrary KEM, it requires certain properties of Diffie-Hellman KEM that simply are not satisfied by any post-quantum candidates. Second, even if adapted to post-quantum KEMs, it would require an ideal permutation with blocklength larger than any known viable candidate.

We show how to modify the RT21 to make it compatible with post-quantum KEM candidates like ML-KEM. We also describe a direct domain-extension construction for ideal permutations, showing how to construct a huge-block ideal permutation directly from one with smaller blocklength, such as the Keccak permutation family. Along the way, we also introduce new abstractions for *oblivious key-value stores* (Garimella et al., Crypto 2021) that make the analysis of these kinds of PSI protocols more modular.

We implemented our protocol and evaluated its performance across different network settings and instantiations. We achieve PSI from standardized post-quantum primitives with latency as low as 0.25 ms/item and communication as low as 1.67 KiB/item. We find that the performance penalty for post-quantum security ranges from 1.25× to 5.92× in latency and is 16.7× in communication, depending on the instantiation.

1 Introduction

Private Set Intersection (PSI) is a fundamental cryptographic primitive that allows two parties to compute the intersection of their respective datasets while revealing nothing about elements outside the intersection. Over the past decade, PSI has evolved from a largely theoretical concept into a practical building block deployed in real-world systems such as contact discovery in messaging platforms and privacy-preserving advertising.

Small vs. Large Sets There are two broad parameter regimes for PSI that affect the protocol design. In **large-set** PSI, the parties have sets containing thousands of items or more. Leading protocols in this regime [36, 35, 24, 33, 9, 34, 19, 37] all use a fixed number (usually 128) of oblivious transfers ("base OTs") that require public-key operations. Then the rest of the protocol, which scales with the number of items, is extremely cheap: usually 10s of bytes of communication and a small number of cheap *symmetric-key* cryptographic operations per item.

How expensive is the fixed overhead in these protocols? For context, 128 base OTs using the protocol of [27] (as a representative example) require generating and sending 128 KEM public keys, generating and sending 256 KEM encapsulations, and finally performing 128 KEM decapsulations. When the PSI input sets are very large, these fixed costs indeed have almost no impact on the overall protocol performance.

However, in the **small-set** PSI regime, when the sets have several hundred items or less, we favor protocols where the number of KEM operations scales exactly with the number of items, with no fixed costs. Such protocols will be cheaper than the fixed base OTs required by the large-set PSI protocols. At this scale, the

number of protocol rounds also has a much larger effect, and small-set PSI protocols can have significantly better round complexity than large-set protocols. For these reasons, the leading protocol for small sets [39] outperforms the leading large-set protocol for sets of size roughly 500 or fewer.

Many practical applications naturally involve small inputs. Examples include: a privacy-preserving alternative to Apple's Airdrop feature [23], privacy-preserving authentication for SSH [40], and PSI between sets with vastly different cardinalities, where the protocol reduces to a PSI instance whose cost is proportional to the smaller set [16]. These use cases highlight the importance of designing PSI protocols that are explicitly optimized for small sets.

PSI Standards in the (Post-Quantum) Future PSI is the cryptographic core of many emerging real-world privacy-preserving systems. The literature on PSI is vast and is converging to a solid level of maturity. In light of these facts, NIST has started to take exploratory steps towards standardizing PSI and PSI-like protocols. Their 2024 Workshop on Privacy-Enhancing Cryptography (WPEC) contained an entire day devoted to PSI [31].

Any potential future standardization efforts are sure to demand post-quantum (PQ) security. Our work is motivated by the following research questions:

What is the performance penalty for making PSI post-quantum?

To what extent can we base PSI on ***standardized*** *post-quantum components?*

For large-set PSI, the answers seem relatively straight-forward. Quantum attacks only affect the fixed setup of base OTs, which use public-key cryptography (usually Diffie-Hellman). By design, replacing the base OTs with post-quantum ones will have a minimal effect on overall performance.

On the other hand, the leading small-set PSI protocols (1) absolutely rely on quantum-vulnerable methods like Diffie-Hellman, and (2) are not so modular that we can easily replace a well-defined component (like OT) with a PQ alternative. Rather, these PSI protocols are tied to Diffie-Hellman techniques in a fundamental way. The PSI literature is largely quiet on the question of how Diffie-Hellman techniques can be replaced with PQ alternatives.

1.1 Our Approach

Starting Point We focus exclusively on PSI protocols that provide active security (i.e., against malicious adversaries).[1] The protocol of Rosulek & Trieu [39] (hereafter "RT21") is the leading such PSI for small sets. It presents the following challenges for post-quantum security:

- It requires a CCA-secure KEM[2] in which valid ciphertexts are indistinguishable from random strings, *even to the secret-key holder!* The only suggested instantiation in RT21 is hashed Diffie-Hellman, which of course is not post-quantum. Unfortunately, no leading PQ KEM candidates, including NIST standard ML-KEM, enjoy this strong pseudorandomness property. They all achieve CCA security using the Fujisaki-Okamoto transformation [18], therefore random strings trigger a decryption error while valid ciphertexts do not.
- It requires an ideal permutation with block-length large enough to fit a KEM ciphertext or public key. For typical DH (over an elliptic curve) this is 256 bits, but for ML-KEM this is 800–1600 *bytes!* It is not obvious how to instantiate such a wide ideal permutation in practice.

Our contributions. (1) We modify the RT21 protocol framework to support modern (FO-based) KEMs. Roughly speaking, in the first protocol message of RT21, Alice sends a single KEM public key. In our modification, Alice must generate n KEM public keys (one for each item in her set). Thus, Alice's first protocol message is longer relative to RT21, by a factor of her set size.

[1] Protocols that are secure only against semi-honest adversaries seem a dubious choice for standardization.

[2] RT21 is written in terms of a 2-round key agreement protocol, but it is a simple matter to write RT21 in terms of KEM syntax instead.

On the plus side, our modification allows the protocol to use only 2 rounds. In RT21, 2 rounds are possible only when KEM encapsulations/ciphertexts do not depend on the public key, which is only known for Diffie-Hellman-style KEMs and not for PQ candidates. Interestingly, our modification also means that we no longer require CCA security from the underlying KEM, although standardized KEMs are of course CCA-secure.

(2) We introduce a *direct* domain-extension construction for ideal permutations: starting from a single n-bit ideal permutation, we construct an N-bit ($N \gg n$) permutation that is indifferentiable from an ideal permutation. Generic domain extension can also be obtained through constructions such as Feistel networks or by composing known transformations through random oracles and ideal ciphers. In contrast, our construction works directly in the ideal-permutation model, making it particularly natural for modern permutation-based cryptography and for public permutations such as Keccak-f. Moreover, as shown in Theorem 2, our analysis gives a concrete distinguishing bound of $O(q^2r^2/2^{n-s})$, providing strong concrete security for the large expansion factors required in our application. This allows us to extend Keccak-f[1600] directly to the block sizes required for ML-KEM public keys and ciphertexts with practical parameters.

We additionally evaluate Feistel-based domain extension as a provable alternative and fixed-key wide-block permutations as heuristic alternatives, allowing us to quantify the performance cost of insisting on an unkeyed permutation with ideal-model security.

(3) RT21 encodes certain values in a polynomial via interpolation. Taking RT21 at face value, to support ML-KEM we would need the polynomial to be over the field $GF(2^{800 \cdot 8})$, which would be prohibitively expensive. However, it is well-known since [19] that the polynomial can be replaced by any "oblivious key-value store" (OKVS) construction.

In rewriting the PSI protocol in terms of arbitrary OKVS, we introduce new abstractions for OKVS (*simulatable* and *extractable* OKVS) that capture RT21's idiom for applying OKVS in a protocol with malicious participants. This improves the modularity of the protocol and may be of independent interest when applying OKVS in other settings. The result of these abstractions is an intuitively simple PSI protocol (e.g., "encode n KEM public keys into a simulatable/extractable OKVS" — see our full protocol in Figure 5).

Security Like RT21, our protocol is UC-secure in the presence of malicious adversaries, and requires a random oracle and ideal permutation. We do not provide an end-to-end security proof for PSI against *quantum* adversaries. Rather, we prove security against classical adversaries, but using post-quantum building blocks.

Implementation & Evaluation We implemented several variants of our protocol and compared them to the state of the art:

- A conservative version of our protocol, with security assuming: ML-KEM-512 is CPA secure, the Keccak permutation is an ideal permutation, and SHA256 is a random oracle.
- A non-PQ version of our protocol, using elliptic curve Diffie-Hellman in place of ML-KEM.
- Less conservative (PQ) versions of our protocol, where we replace Keccak (the overwhelming performance bottleneck above) with different permutations, some only heuristic.
- The leading small-set, *non-PQ* PSI [39].
- The leading large-set PSI [37], instantiated with PQ base OTs.

We summarize our findings below:

- Unsurprisingly, our protocol requires significantly more communication than non-PQ alternatives. In our experiments, the communication cost increases by approximately 16×–17× compared to our non-PQ baseline, which is consistent with the larger ciphertext sizes of post-quantum KEMs. Compared to [39], our protocol incurs approximately 26×–28× more communication across all set sizes.

- The conservative version of our protocol is significantly slower than non-PQ PSI. The bottleneck is not the KEM computation (ML-KEM has comparable efficiency to Diffie-Hellman), but our extensive use of the Keccak permutation (24 and 12 invocations per item). Keccak is relatively slow, and its inverse, which we also require, is even slower. Compared to [39], the 24-round Keccak-1600 instantiation is about 1.8×–8.7× slower across our measured settings, with the largest slowdown appearing in the WAN single-thread setting. The 12-round Keccak instantiation improves this, but is still about 1.4×–6.4× slower than RT21.
- Different instantiations of the ideal permutation are much faster. In particular, replacing the 24-round Keccak-1600 with more efficient designs significantly reduces the overhead. 12-round Keccak-1600 reduces the runtime to about 60%–80% of the full-round version, while constructions such as HCTR2, SNEIK and Feistel further reduce the gap and can achieve performance close to the non-PQ baseline in LAN settings (within about 0.9×–1.8×), while incurring a 1.1×–4.0× slowdown in WAN settings due to increased communication costs.
- On a fast network, our SNEIK instantiation is faster than the leading PQ large-set PSI for $n \leq 2^8$. On a slow network, it remains faster for $n \leq 2^9$. Meanwhile, the 24-round Keccak-1600 instantiation is faster than the leading PQ large-set PSI on a fast network for $n \leq 2^7$ when running in multi-thread mode, and on a slow network for $n \leq 2^8$ in all modes.

1.2 Other Closely Related Work

RT21 has an acknowledged similarity to the PSI framework of Choi, Dachman-Soled, and Jarecki [11] (CDJ16). In some ways, our new variant of RT21 has even more similarities to CDJ16. We discuss the similarities in more detail later in Section 5 after describing our PSI protocol.

Generic domain extension can be obtained by combining known constructions through random oracles and ideal ciphers. Our construction differs in that it directly promotes a small-block ideal permutation into a large-block ideal permutation. Chakraborty, Dutta, and Kundu [8] use a small-block ideal permutation to construct a wide-block (keyed) SPRP. In other words, the construction is proven to be an SPRP and not indifferentiable from an ideal permutation. Other work on (variable-width) wide-block SPRPs does not build directly on ideal permutations.

2 Preliminaries

2.1 Private Set Intersection (PSI)

Figure 1 describes the ideal functionality for private set intersection (PSI) that we use. We point out several important features of the functionality: (1) Only one party, Alice, receives output. The other party receives nothing. (2) It is assumed that parties know (an upper bound) on the size of the input sets, since they are public parameters of the functionality. Consequently, the protocol is allowed leak this cardinality information. (3) The functionality allows corrupt parties to provide input sets that are "larger than advertised," but only up to a limit cardinality $\widetilde{n}$. Our protocol achieves a particular relationship between n and $\widetilde{n}$ that is described later.

We define security of MPC protocols in the universal composition (UC) model of Canetti [7].

2.2 Key Encapsulation Mechanisms (KEM)

We adopt the definition of a key encapsulation mechanism (KEM) consistent with recent NIST standardization efforts [1].

Definition 1. *A **key encapsulation mechanism (KEM)** is a triple of probabilistic polynomial-time algorithms* (KeyGen, *Encaps*, *Decaps*) *defined as follows:*

Public Parameters:
$\mathcal{U}$: universe of items
n_A: maximum cardinality of honest Alice's set
n_B: maximum cardinality of honest Bob's set
$\widetilde{n}_A$: maximum cardinality of *corrupt* Alice's set
$\widetilde{n}_B$: maximum cardinality of *corrupt* Bob's set

Behavior:
await input $X \subseteq \mathcal{U}$ from Alice
  abort if Alice is honest and $|X| > n_A$
  abort if Alice is corrupt and $|X| > \widetilde{n}_A$

send `ready` to Bob

await input $Y \subseteq \mathcal{U}$ from Bob
  abort if Bob is honest and $|Y| > n_B$
  abort if Bob is corrupt and $|Y| > \widetilde{n}_B$

send $X \cap Y$ to Alice

Figure 1: $\mathcal{F}_{\text{psi}}$: ideal functionality for PSI

- $\mathsf{KeyGen}(\lambda)$*: On input the security parameter, outputs a key pair* (sk, pk)*.*
- $\mathit{Encaps}(pk)$*: A randomized algorithm, that on input a public key* pk *outputs a pair* (c, k)*, where* $c \in \mathcal{C}_\lambda$ *is a ciphertext and* k *is a shared secret.*
- $\mathit{Decaps}(sk, c)$*: On input a secret key* sk *and a ciphertext* c*, outputs* k*, which may equal an error indicator symbol* $\perp$*.*

We write $\mathcal{C}_\lambda$ *and* $\mathcal{P}_\lambda$ *for the space of possible ciphertexts and public keys, respectively.*

Interestingly, our PSI protocol does not require the traditional correctness property for a KEM. This is because the protocol does not use the k values anywhere! Instead, it only checks whether $\mathsf{Decaps}(sk, c) \neq \perp$ as a simple "liveness" check. And despite targeting security against malicious adversaries, we do not require a CCA-secure KEM. Instead, we require only the following properties:

Definition 2 ([20])**.** *A KEM has* ***pseudorandom public keys*** *if the following two distributions are indistinguishable:*

$(-, pk) \leftarrow \mathsf{KeyGen}(\lambda)$ return pk	$pk \leftarrow \mathcal{P}_\lambda$ return pk

Definition 3 ([20])**.** *A KEM has* ***pseudorandom ciphertexts*** *if the following two distributions are indistinguishable:*

$(-, pk) \leftarrow \mathsf{KeyGen}(\lambda)$ $(c, -) = \mathsf{Encaps}(pk)$ return (pk, c)	$(-, pk) \leftarrow \mathsf{KeyGen}(\lambda)$ $c \leftarrow \mathcal{C}_\lambda$ return (pk, c)

Definition 4. *A KEM has* ***sparse ciphertexts*** *if the following program outputs true with only negligible probability:*

$(sk, pk) \leftarrow \mathsf{KeyGen}(\lambda)$
$c \leftarrow \mathcal{C}_\lambda$
return $[\mathsf{Decaps}(sk, c) \stackrel{?}{\neq} \perp]$

Instantiations. The broad goal of our work is to instantiate all ingredients necessary for PSI using post-quantum, and ideally standardized constructions. Lattice-based standards like ML-KEM do not immediately satisfy these definitions, but can be easily adapted in the following ways:

- Günther, Stebila, and Veitch [20] introduce the *obfuscated KEM* (OKEM) framework, which is a wrapper around a traditional KEM that encodes public keys and ciphertexts (in the case of ML-KEM, these are vectors over $\mathbb{Z}_q$) into pseudorandom *bitstrings*. The encoding procedure is called Kemeleon, and when it is composed with the ML-KEM construction in the natural way, we obtain a KEM satisfying our definitions 2 and 3 (under the mild assumption that ML-KEM is only *CPA-secure*). Thus, the space of keys and ciphertexts $\mathcal{K}_\lambda$ and $\mathcal{C}_\lambda$ are sets of bitstrings of an appropriate length.
- The ML-KEM standard uses an *implicit-rejection* variant of the Fujisaki-Okamoto transformation. If the ciphertext is not well-formed, then Decaps returns a random output. Our definition 4 demands that the Decaps algorithm frequently returns $\perp$. This definition can be satisfied by using an *explicit-rejection* variant of ML-KEM, in which Decaps returns $\perp$ on malformed ciphertexts.[3]

For purposes of comparison, we also consider a non-post-quantum KEM. Plain ECDH does not satisfy the sparse ciphertexts property (Decaps never returns $\perp$). A simple way to gain sparsity is to add some redundancy to ECDH. Let public keys have the form $pk = g^{sk}$ and let ciphertexts have the form $(g^r, H(pk^r))$ where H is a simple key derivation function with λ bits of output. Decapsulation of (R, h) returns $\perp$ unless $h = H(R^{sk})$. To achieve the necessary pseudorandomness properties, one must also apply the Elligator encoding/resampling technique of [4]. Assuming a 256-bit curve and 128-bit function H, the resulting KEM public keys have 256 bits and ciphertexts have 384 bits.

2.3 Indifferentiability

Maurer, Renner, and Holenstein introduced the indifferentiability framework [28] for proving security of *unkeyed* constructions like random oracles, ideal ciphers, etc. We use this framework to establish the security of our ideal-permutation domain extension construction.

Definition 5. *Let $\mathcal{P}$ ("primitive") and $\mathcal{T}$ ("target") be ideal oracles.*[4] *Let $\mathsf{Cons}^{\mathcal{P}}$ be a construction in the $\mathcal{P}$-ideal world that provides the same interface as a $\mathcal{T}$ oracle. We say that Cons is **indifferentiable** from $\mathcal{T}$ if there exists an efficient simulator Sim such that the following pairs of oracles are indistinguishable:*

$$(\mathsf{Cons}^{\mathcal{P}}, \mathcal{P}); \quad (\mathcal{T}, \mathsf{Sim}^{\mathcal{T}})$$

Intuitively, using a public oracle $\mathcal{P}$ to instantiate Cons is equivalent to having access to an ideal $\mathcal{T}$ oracle. While indifferentiability does not compose with all possible security definitions [38], it is sufficient for our purposes.[5]

3 Oblivious Key-Value Stores

The RT21 protocol encodes certain values together into a polynomial, using interpolation. Later, Garimella et al. introduced a generalization of this polynomial encoding technique, called **oblivious key-value stores** (OKVS) in [19]. Our PSI protocol is written in terms of the OKVS abstraction. Below we review the standard definitions of OKVS before introducing some new definitions/abstractions.

[3] If one insists on using the (implicit-rejection) ML-KEM standard as a black-box, then we can still obtain the needed sparsity property by applying the transformation described immediately below, which increases the ciphertext size by, e.g., 128 bits.

[4] In our construction, $\mathcal{P}$ is a small-block ideal permutation and $\mathcal{T}$ is a wide-block ideal permutation.

[5] All of our security definitions involve *single-stage* games, in the terminology of [38].

3.1 Plain OKVS

We recall the definitions of OKVS from [19]. Our results only require a *linear* OKVS, so we present definitions that are specialized for that case.

Definition 6. *An **oblivious key-value store (OKVS)** consists of a pair of algorithms* $(\mathsf{Encode}, \mathsf{Decode})$*, and parameters* $\mathcal{K}$*,* $\mathcal{V}$ *(sets), and cardinality bound* n*. It must satisfy the following properties:*

- ***Correctness:*** *For all* $S = \{(k_1, v_1), \ldots, (k_m, v_m)\} \subseteq \mathcal{K} \times \mathcal{V}$*, where the* k_i*'s are distinct and* $m \leq n$*,*

$$\Pr[\mathsf{Decode}(\mathsf{Encode}(S), k_i) \neq v_i]$$

  is negligible (over the probability in Encode*).*

- ***(Strong) obliviousness:*** *Then for all* $\{k_1, \ldots, k_m\} \subseteq \mathcal{K}$ *and* $\{v_1, \ldots, v_m\} \subseteq \mathcal{V} \cup \{\perp\}$*, with* $m \leq n$*, the following two distributions are indistinguishable:*

for $i = 1$ to m:
  if $v_i = \perp$: $v_i \leftarrow \mathcal{V}$
  $S = S \cup \{(k_i, v_i)\}$
return $\mathsf{Encode}(S)$
	for $i = 1$ to m:
	  if $v_i \neq \perp$:
	  $S = S \cup \{(k_i, v_i)\}$
	return $\mathsf{Encode}(S)$

  In other words, it makes no difference to associate key k *with a uniformly chosen value* v*, or to exclude* k *from* Encode*'s argument altogether.*

Obliviousness implies that the output of Encode hides all keys k_i whose corresponding value v_i is chosen uniformly.

Definition 7. *An OKVS is **linear** if:*

- *The output of* Encode *is an element of* $\mathbb{F}^\ell$ *for some field* $\mathbb{F}$*.*
- *There is a function* $\mathsf{Probe} : \mathcal{K} \to \mathbb{F}^\ell$ *such that* Decode *can be written as the inner product* $\mathsf{Decode}(D, k) = \langle D, \mathsf{Probe}(k) \rangle$*.*
- $\mathsf{Encode}(S)$ *function produces a **random** solution* D *to the system of linear equations* $\{\mathsf{Decode}(D, k) = v \mid (k, v) \in S\}$*.*

Note: The correctness property for a linear OKVS is equivalent to the following: for all $\{k_1, \ldots, k_m\} \subseteq \mathcal{K}$ with $m \leq n$, the probe vectors $\{\mathsf{Probe}(k_1), \ldots, \mathsf{Probe}(k_m)\}$ are linearly independent except with negligible probability. Additionally, correctness for a linear OKVS implies (strong) obliviousness, since we insist that Encode samples a uniform solution to the set of linear constraints.

The canonical example of an OKVS is polynomial-OKVS, where $\mathcal{K} = \mathcal{V} = \mathbb{F}$ is a field, and $\mathsf{Encode}(S)$ interpolates a polynomial through the points S. Polynomial-OKVS is a linear OKVS. We discuss other constructions in Section 3.4.

3.2 Simulatable and Extractable OKVS

The RT21 protocol uses a polynomial-OKVS, but composes it with a random oracle and ideal permutation in a special way to achieve security in the presence of a malicious adversary. We introduce stronger security definitions for OKVS that capture the security of this idiom. We present two definitions, separately capturing security when the adversary generates the OKVS (extractability) and when it receives an OKVS (simulatability).

These definitions require the OKVS to exist in an idealized model (e.g., the ideal permutation model), so we explicitly extend the $\mathsf{Encode}, \mathsf{Decode}$ algorithms to use an idealized oracle $\mathcal{O}$.

Simulatability: When an honest party generates an OKVS D, it has "control" over $\mathsf{Decode}^{\mathcal{O}}(D,k)$ for only a bounded number of keys k. Intuitively, our first definition states that a simulator (by playing the role of the oracle $\mathcal{O}$) can generate a special OKVS D^* such that it controls $\mathsf{Decode}^{\mathcal{O}}(D^*,k)$ for *every* k.

Definition 8. *An OKVS is* ***simulatable*** *if there exists simulator algorithms* $(\mathsf{SimOracle}, \mathsf{SimOKVS})$ *such that the following games are indistinguishable:*

adversary has access to honest $\mathcal{O}$ oracle receive S from adversary $D = \mathsf{Encode}^{\mathcal{O}}(S)$ give D to adversary give adversary the following oracle: $\underline{\mathsf{Test}(k)}$: return 1	adversary has access to $\mathsf{SimOracle}$ oracle (as $\mathcal{O}$) receive S from adversary KV = random oracle $\mathcal{K} \to \mathcal{V}$, except $\forall (k,v) \in S$: $\mathsf{KV}(k) = v$ $D = \mathsf{SimOKVS}^{\mathsf{KV}}(S)$ give D to adversary give adversary the following oracle: $\underline{\mathsf{Test}(k)}$: return $\mathsf{Decode}^{\mathcal{O}}(D,k) \stackrel{?}{=} \mathsf{KV}(k)$

In the right-hand game, we allow $\mathsf{SimOracle}$ *and* $\mathsf{SimOKVS}$ *to be stateful and share state. In particular,* $\mathsf{SimOracle}$ *may access the oracle* KV *that was provided to* $\mathsf{SimOKVS}$ *(of course, only after* $\mathsf{SimOKVS}$ *was called).*

This definition demands that the simulator can cause $\mathsf{Decode}(D,\cdot)$ to agree with an entire *externally supplied* key-value mapping KV. Imagine that KV samples its outputs with known trapdoor (indeed, this is how this security property is used in the PSI proof); this means that the simulator can know a trapdoor for *every* $\mathsf{Decode}(D,k)$.

Extractability Obliviousness means that (under normal circumstances) the OKVS output hides the choice of keys used for encoding. Our second definition demands that the simulator can extract these set of keys from an adversarially generated OKVS.

Definition 9. *An OKVS is* β***-extractable*** *if there exists simulator algorithms* $(\mathsf{ExtOracle}, \mathsf{ExtOKVS})$ *such that the following games are indistinguishable:*

adversary has access to honest $\mathcal{O}$ oracle receive D from adversary give adversary the following oracle: $\underline{\mathsf{Test}(k)}$: return 1	$\underline{\mathsf{Samp}()}$: $r \leftarrow \mathcal{U}$ $\mathcal{Q} = \mathcal{Q} \cup \{r\}$ return r adversary has access to $\mathsf{ExtOracle}^{\mathsf{Samp}}$ oracle receive D from adversary $\widetilde{K} = \mathsf{ExtOKVS}(D)$ give adversary the following oracle: $\underline{\mathsf{Test}(k)}$: $v = \mathsf{Decode}^{\mathcal{O}}(D,k)$ return $k \in \widetilde{K} \vee v \in \mathcal{Q}$

Additionally, the output of ExtOKVS *must be a set of cardinality at most* $\beta(n)$*, where* n *is the cardinality parameter of the OKVS.*

As above, in the right-hand game we allow ExtOracle *and* ExtOKVS *to share state. In particular,* ExtOKVS *may know how the adversary has queried its oracle (played by* ExtOracle*).*

The definition may require some effort to fully understand. By asking for the two games to be indistinguishable, we are demanding that it is negligibly likely to find an input that causes Test to output 0. Thus, for all $(k, v = \mathsf{Decode}(D, k))$, one of two cases must hold:

- $k \in \widetilde{K}$: The extraction algorithm has identified k as one of the special keys it used to generate D. Intuitively, the adversary may have chosen v.
- $v \in \mathcal{Q}$: This value was chosen internally by the Samp subroutine. Intuitively, the adversary did not choose v.

Thus, the extractor ExtOKVS identifies a small (at most $\beta(n)$ items) set of keys for which the adversary controlled $\mathsf{Decode}(D, k)$, while *all other* outputs of $\mathsf{Decode}(D, \cdot)$ are beyond the adversary's control.

3.3 Construction

We now show how to transform a plain OKVS into a simulatable/extractable OKVS, using an ideal permutation and random oracle. This formalizes and modularizes the idiom used by RT21.

Theorem 1. *Let* $(\mathsf{Encode}, \mathsf{Decode})$ *be a linear OKVS supporting with values from field* $\mathcal{V} = \mathbb{F}$*, with* $|\mathbb{F}| \geq 2^{2\lambda}$*. Let* $\ell(n)$ *be the length of the vector output by* Encode *(where* n *is the OKVS's cardinality parameter). Let* $\Pi^{\pm} : \mathbb{F} \to \mathbb{F}$ *be an ideal permutation and let* H *be a random oracle whose range is* $\mathbb{F}$*. Then the following is a simulatable and* β*-extractable OKVS:*

$$\underline{\widetilde{\mathsf{Encode}}^{\Pi^{\pm},H}(S)\text{:}}$$
$$\text{return } \mathsf{Encode}\Big(\Big\{\big(k,\ \ \Pi(v) + H(k)\big) \ \Big|\ (k,v) \in S\Big\}\Big)$$

$$\underline{\widetilde{\mathsf{Decode}}^{\Pi^{\pm},H}(D,k)\text{:}}$$
$$\text{return } \Pi^{-1}\Big(\mathsf{Decode}(D,k) - H(k)\Big)$$

Against an adversary who makes q *queries to* H*, the construction is* β*-extractable for*

$$\beta(n) = \ell(n)\frac{\log|\mathcal{V}| + \kappa/\ell(n)}{\log|\mathcal{V}| - 2\log q},$$

except with probability $1/2^{\kappa}$*.*

Proof. We provide only a sketch here, with the details deferred to Appendix A.

For the proof of simulatability, the simulator is given an oracle KV and produces D; by playing the role of oracles $(\Pi^{\pm}, H)$, it must cause

$$\widetilde{\mathsf{Decode}}(D,\cdot) = \Pi^{-1}\big(\mathsf{Decode}(D,\cdot) - H(\cdot)\big)$$

to agree with $\mathsf{KV}(\cdot)$. It uses the following strategy: whenever the adversary queries $H(k)$, it programs Π at

$$\Pi(\mathsf{KV}(k)) = \mathsf{Decode}(D,k) - H(k).$$

One can easily check that $\widetilde{\mathsf{Decode}}$ indeed agrees with KV, as long as this programming succeeds (i.e., Π was not already programmed at this point). But since $\mathsf{KV}(k)$ and $H(k)$ are chosen randomly at the time of programming, the programming fails with only negligible probability.

For the proof of extractability, the simulator must simultaneously extract from an adversarially generated OKVS, while simulating the oracles $(\Pi^{\pm}, H)$ to ensure that all *other* OKVS outputs are generated by a subroutine Samp. The main idea is that the output of $\widetilde{\mathsf{Decode}}(D, k)$ is an output of Π^{-1}. We can partition all pairs $(x, y = \Pi(x))$ based on whether they are the result of a fresh $\Pi(x)$ query or a fresh $\Pi^{-1}(y)$ query. Only in the former case could x (the output of a Π^{-1} query) be chosen by the adversary, because the simulator can arrange so that x in the latter case was sampled by Samp. Intuitively, the adversary "controlled" $\widetilde{\mathsf{Decode}}(D, k) = v$ is if (1) it queried $H(k)$, and (2) it made v as a *fresh* forward query to Π.

Then it remains to argue that the number of items extracted in this way is bounded by the given expression β. We prove this using a compression argument: We show that if the adversary "controlled" $\widetilde{\mathsf{Decode}}(D, k)$ for some k, then it is possible to use D to deduce $H(k)$ (if also given a very short additional "hint"). Thus, the OKVS D encodes $H(k)$ for all k output by the extraction algorithm. Since H is a random oracle and D has bounded size, this places a bound on the number of items output by the extraction algorithm. □

Comparison to RT21. The construction is slightly different than the idiom used in RT21. In our construction, the underlying plain OKVS is invoked with key-value pairs of the form $(k, \Pi(v) + H(k))$. In RT21, the pairs have the form $(H(k), \Pi(v))$.

Our approach has a slight advantage in certain situations when the k's are short (in the eventual PSI protocol, the k's are the elements of the parties' input sets). Our approach does not require the underlying OKVS to support long keys (e.g., 2λ, the output length of H). For example, if the k values are 64 bits and the underlying OKVS is polynomial-OKVS, we can use polynomial interpolation over $GF(2^{64})$ whereas the RT21 approach demands $GF(2^{256})$. Our analysis of the extraction slack β (maximum number of items extracted by $\mathsf{ExtOKVS}$ from an adversarially generated OKVS) is also tighter than that of RT21.

Both our bound and that of RT21 are sensitive to the size of the values being encoded in the OKVS ($\mathcal{V}$ in our notation). Thus, because ML-KEM keys and ciphertexts are so much bigger than ECDH ciphertexts, it is significantly harder for the adversary to "overfit" their OKVS and have a larger effective PSI input. This is evident in Figure 2, where the adversary's relative advantage over honest parties β/n decreases as the items ($\mathcal{V}$) increase.

3.4 Instantiations

Polynomial-OKVS is linear and has optimal size (n items are encoded into a vector of length n). However, polynomial interpolation is rather slow in practice. A more favorable alternative is the **RB-OKVS** (for "random-band") of [6]. It enjoys extremely fast encoding/decoding, and its expansion factor is close to optimal: 1.07 – 1.16 for the set sizes we consider.

In our implementation, we apply the construction in Theorem 1 to RB-OKVS. Figure 2 shows the resulting concrete parameters. We determine the relevant RB-OKVS parameters using the methodology described in [6]. There are many choices of parameters that would work, with a tradeoff between encoding time and encoding size. We chose a reasonable middle ground (in the end, the choice of OKVS parameters has minimal effect on the PSI protocol's performance).

4 Domain Extension for Ideal Permutations

The construction in Theorem 1 requires an ideal permutation over $\mathcal{V}$, the set of OKVS "values." In our eventual PSI protocol, these values are (encodings of) KEM ciphertexts and public keys. Standardized variants of ML-KEM have ciphertexts of 768, 1088, and 1568 *bytes,* and public keys of 800, 1184, and 1568 bytes. We are not aware of any unkeyed permutations proposed with such wide block length. This section describes approaches for instantiating an ideal permutation of this size.

KEM	$\lvert\mathcal{V}\rvert$	n	ℓ	β	β/n
ECDH + hash	2^{384}	128	144	247	1.93
		256	274	469	1.83
		512	548	939	1.83
		1024	1096	1879	1.83
ML-KEM-512	$2^{768\cdot 8}$	128	144	147	1.15
		256	274	281	1.10
		512	548	562	1.10
		1024	1096	1125	1.10

Figure 2: Concrete OKVS parameters. For the given values of n, the corresponding RB-OKVS encoding has length ℓ to limit the failure probability to 2^{-40}. Then applying Theorem 1, an adversary making 2^{80} oracle queries cannot encode more than β items into the OKVS, except with probability 2^{-64}. In other words, β is the adversary's maximum effective input size in our PSI protocol, which is more than an honest party by a factor of β/n.

4.1 Possible Instantiations

Feistel construction. The Feistel construction is a standard approach to convert a non-invertible primitive into an invertible one. In the unkeyed setting, one may take independent (length-preserving) n-bit random oracles as the Feistel round functions, resulting in an unkeyed permutation on $2n$ bits. It is known that 8 rounds suffice for the result to be indifferentiable from an ideal permutation [15]. Unfortunately, the security bound of this construction is poor: $O(q^8/2^n)$. This security level may suffice for large enough blocklength n (i.e., $n \geq 8\lambda$), but it is not a suitable solution for all practical parameter ranges.

A heuristic solution (that fails). One approach is to start from a *keyed* permutation (i.e., strong PRP). The leading such candidate is HCTR [43] and its derivatives [13], which have been selected for NIST standardization as a so-called *accordion mode* of operation. Thus, a heuristic approach may be to use HCTR with a fixed (and public) key, as an ideal permutation. This is analogous to using fixed-key AES as a 128-bit ideal permutation — a common practice in MPC implementations [2, 21].

Unfortunately, fixed-key HCTR fails to be an ideal permutation, even if its components are ideal. In Appendix B we show how to efficiently find input-output pairs that have a certain correlation that is not feasible to find in an ideal permutation. We are not sure whether this observation can be leveraged for a concrete attack on our final PSI protocol, only that fixed-key HCTR fails to meet the conditions needed by our security proof.

4.2 Our Construction

We describe a novel *direct* domain-extension construction for ideal permutations. Starting from an n-bit ideal permutation, we construct an N-bit ($N \gg n$) permutation that is indifferentiable from an ideal permutation and can serve as the "ultra-wide" ideal permutation required to encode ML-KEM ciphertexts and public keys (e.g., 1568 bytes = 12544 bits). Unlike the Feistel construction, our construction is built from an underlying *invertible* primitive. Additionally, our construction has a significantly tighter security bound.

The construction Our construction is given in Figure 3 and illustrated in Figure 4. We use lowercase letters to denote n-bit strings (and functions with n-bit input and output), and uppercase letters to denote N-bit strings (and functions with N-bit input and output).

Assuming access to an ideal n-bit permutation $\pi^{\pm}$, we construct an N-bit permutation $\Pi^{\pm}$. After initializing the internal state with the N-bit input, the construction performs many rounds of the following basic steps: (1) apply π to the n-bit prefix of the current state; (2) cyclic shift the current state left by $s < n$

Parameters:
n: blocklength of $\pi^{\pm}$
N: blocklength of the construction $\Pi^{\pm}$
s: shift amount, such that $s \geq 2\lambda$ and $n - s \geq 2\lambda$
t: such that $N = n + s(t-1)$
r: number of rounds (calls to π) $= 5t - 2$

$\Pi(X[1..N])$:
  for $i = 1$ to $r - 1$:
    $X[1..n] := \pi(X[1..n])$
    $X[s+1..n] := X[s+1..n] \oplus i$
    cyclic shift X left by s bits
  $X[1..n] := \pi(X[1..n])$
  return $X = X[1..N]$

$\Pi^{-1}(X[1..N])$:
  $X[1..n] := \pi^{-1}(X[1..n])$
  for $i = r - 1$ down to 1:
    cyclic shift X right by s bits
    $X[s+1..n] := X[s+1..n] \oplus i$
    $X[1..n] := \pi^{-1}(X[1..n])$
  return $X = X[1..N]$

Figure 3: Our construction of an N-bit permutation $\Pi^{\pm}$ from an n-bit permutation $\pi^{\pm}$.

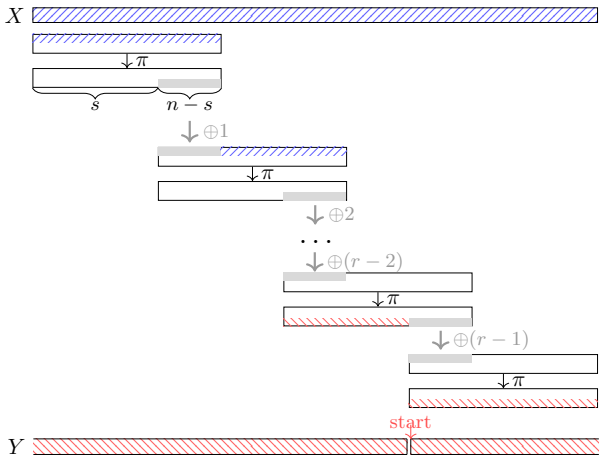

Figure 4: Illustration of the construction.

bits; (3) xor the prefix of the current state with the round number. After a certain number of rounds, the final state is taken as the output.

The state is shifted by $s < n$ bits so that $n - s$ bits "overlap" between calls to π (see the figure). Define an integer $t = 1 + (N - n)/s$; thus, in t consecutive rounds, exactly N bits of the state have been "touched" (as input or output to π), and t consecutive calls to π uniquely determine the construction's internal state. For such a definition of t, our construction requires $r = 5t - 2$ rounds total, meaning that it makes roughly 5 passes over the input. For technical reasons, $s \geq 2\lambda$ and $n - s \geq 2\lambda$, meaning $n \geq 4\lambda$.

Theorem 2. *If π is an ideal n-bit permutation, then Π is indifferentiable from an ideal N-bit permutation with distinguishing advantage bounded by $O(q^2r^2/2^{n-s})$.*

Proof. We provide a summary here and defer the full proof to Appendix D.

In the indifferentiability framework, we consider a real world in which the adversary has oracle access to our construction and to $\pi^{\pm}$, then an ideal world in which the adversary has oracle access to an *ideal* $\Pi^{\pm}$ permutation and a simulator (who itself has oracle access to $\Pi^{\pm}$). The simulator's goal is to program $\pi^{\pm}$, making it appear as though the ideal $\Pi^{\pm}$ arose from running our construction on $\pi^{\pm}$.

Let us define a graph whose nodes are pairs (x, y) where the adversary has either made the query $\pi(x) \to y$ or $\pi^{-1}(y) \to x$. Note that x, y are n-bit strings. Draw an edge $(x, y) \xrightarrow{i} (x', y')$ if $y[s+1:n] \oplus x'[1:n-s] = i$.

The meaning of the edge is that "there exists an execution of our construction that queries $\pi(x) \to y$ in round i and $\pi(x') \to y$ in round $i+1$."

By the definition of parameter t, any path in this graph consisting of t nodes, whose edge labels are *consecutive* integers, uniquely determines the entire state of the construction at some moment. We call such a path a **completable** path. A natural simulation approach is the following:

Observe the adversary's queries to $\pi^{\pm}$ while maintaining this graph. When a completable path P appears, we can compute the entire internal state Z of the construction. Suppose the midpoint edge of P has label less than $r/2$ (the path is "closer" to round 1 than to round r). Then starting from intermediate state Z, we can run the construction backwards (i.e., via our Π^{-1} algorithm) until we have the initial state X. Then we can query our ideal Π oracle to learn $Y = \Pi(X)$. Finally, we can program π so that running the construction forwards from Z results in final state Y. We can successfully program π if we can append at least t "fresh" nodes to the path before reaching the end of the construction, because t nodes are enough to uniquely determine the final state Y.

Intuitively, this approach works as long as the number of rounds is $r \geq 3t$: Starting from round r, we need space for t "fresh" nodes (that the simulator can program) before reaching the end of P, then t nodes for P, then at most t nodes until reaching round 1 (because the assumption is that P is "closer to round 1").

While this strategy captures the main intuition, it doesn't quite work. The reason is that the simulator "completes" the path by (in part) running the construction in its head, thereby adding nodes and edges to the graph. But these nodes and edges can cause new paths to be completable. Unfortunately, it is possible to cause the simulator to enter an infinite loop trying to service all the completable paths that appear.

We avoid this problem by making a distinction between nodes that arise from the adversary's queries to π (we call these **visible** nodes), and nodes arising from the simulator completing paths (invisible nodes). The simulator will only take action when it sees a completable path consisting *entirely of visible nodes.* By definition, completing these paths does not introduce any visible nodes, and avoids any recursion.

We show that a visibly-completable path can be followed/preceded by at most $t-1$ consecutive invisible nodes. As such, our simulation strategy works if the number of rounds is $r \geq 5t-2$, from the following reasoning: Taking the previous example, starting at the last round and working backwards, we need t completely "fresh" nodes on which the simulator can program, then there can be at most $t-1$ invisible nodes before reaching P (t nodes), then at most $2t-1$ more rounds before reaching round 1 (as above, because P is "closer to round 1" than round r).

This summary omits some other cases that the simulator must handle, but it captures the main technique as well as the reason for $r = 5t-2$ rounds. □

Domain separation. Our eventual PSI protocol requires each party to generate an OKVS; each OKVS therefore requires an independent instance of its underlying ideal permutation. Our construction can be modified in the following simple way to implement two independent ideal permutations.

In our construction, the current round number i is xor'ed into the current state. We can implement independent permutations $\Pi_0^{\pm}, \Pi_1^{\pm}$ by defining $\Pi_b^{\pm}$ to xor $i \| b$ into the state. The reason this works is that the entire simulation strategy is defined in terms of a graph, whose edges are determined by what value is xor'ed into the state. Since the two permutations xor disjoint values into the state, the two simulations involve graphs with completely disjoint edges. Essentially, this prevents the two simulation strategies from interfering.

5 The PSI Protocol

We present our PSI protocol in Figure 5. The main idea is as follows:

- Alice creates an OKVS that maps each item in her set to an independent KEM public key. She sends the OKVS to Bob.

- For each item in his set, Bob decodes the associated public key from the OKVS and encapsulates a KEM ciphertext. He encodes an OKVS that maps each item in his set to the corresponding ciphertext, and sends the OKVS to Alice.
- For each of Alice's items, she decodes the associated KEM ciphertext from Bob's OKVS and attempts to decapsulate it. If decapsulation succeeds, she includes that item in the final output.

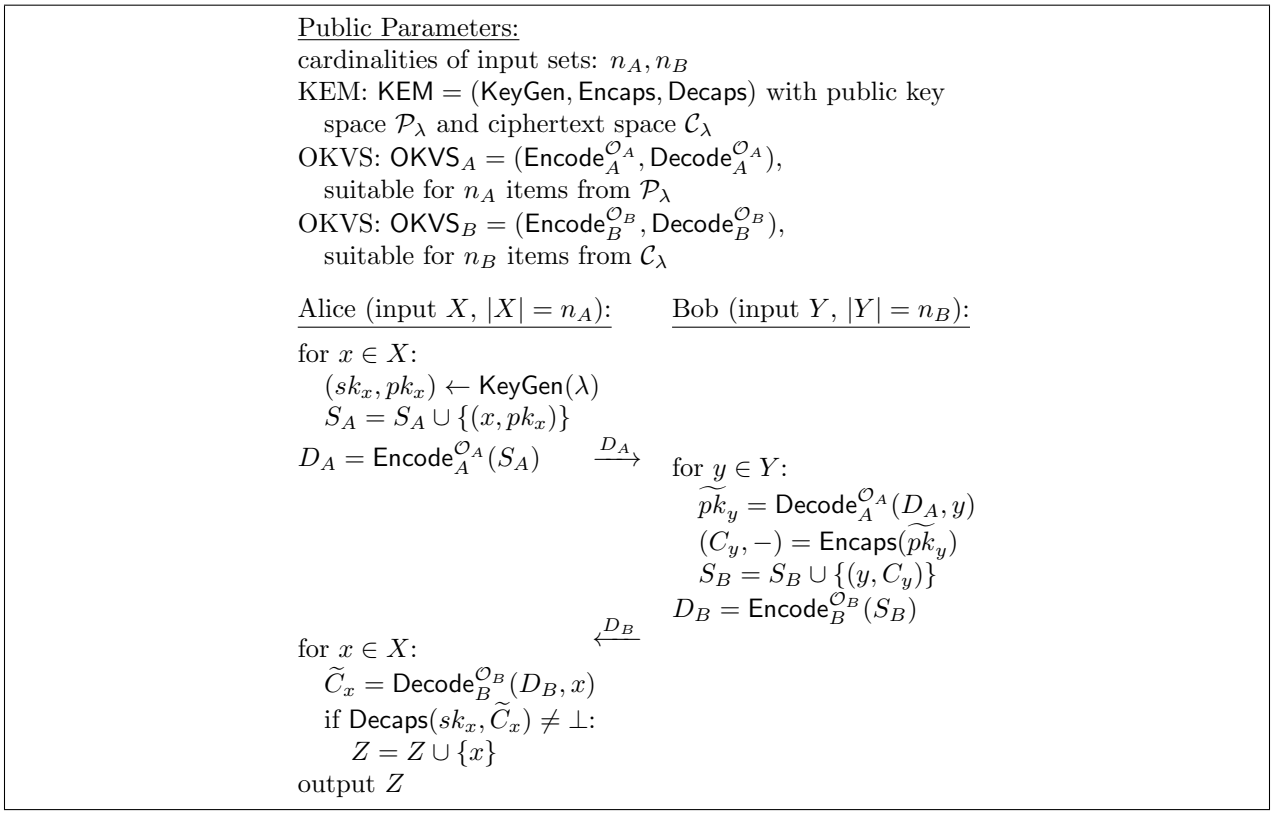

Figure 5: Our PSI Protocol

Theorem 3. *The protocol in Figure 5 UC-securely realizes $\mathcal{F}_{psi}$ (Figure 1), if $\mathsf{OKVS}_A, \mathsf{OKVS}_B$ are simulatable (Definition 8) and extractable (Definition 9), and KEM has pseudorandom public keys (Definition 2), pseudorandom ciphertexts (Definition 3), and sparse ciphertexts (Definition 4).*

The PSI protocol realizes $\mathcal{F}_{psi}$ with parameters $\widetilde{n}_A = \beta(n_A)$, $\widetilde{n}_B = \beta(n_B)$ (cardinality bounds for corrupt parties), where β is the parameter from the definition of OKVS extractability.

Proof intuition. Suppose Alice is corrupt. She begins the protocol by sending her OKVS D_A. If the OKVS is extractable, the simulator can extract a small set of candidate values $\widetilde{X}$, such that for all $x' \notin \widetilde{X}$ Alice had no control over $\widetilde{pk}_{x'} = \mathsf{Decode}_A(D_A, x')$. As such, she cannot distinguish valid KEM ciphertexts under this public key from random strings. Hence, she cannot learn whether Bob holds x' (causing $\mathsf{Decode}_B(D_B, x')$ to be a valid ciphertext) or not.

Suppose Bob is corrupt. Then the simulator can generate a simulated OKVS D_A from Alice, such that it knows the secret key corresponding to *every* public key $\widetilde{pk}_y = \mathsf{Decode}(D_A, y)$. When Bob sends his OKVS D_B, the simulator can also extract a small set of candidates $\widetilde{Y}$, such that Bob has no control over $\mathsf{Decode}(D_B, y')$ for all $y' \notin \widetilde{Y}$. Now,

- For $y' \notin \widetilde{Y}$, since Bob cannot control $\widetilde{C}_{y'} = \mathsf{Decode}(D_B, y')$, it is negligibly likely that this ciphertext will decapsulate correctly. Thus, Alice will never include y' in her output; in other words, her protocol output is always a subset of $X \cap \widetilde{Y}$.
- For $y' \in \widetilde{Y}$, the simulator can check if $\widetilde{C}_{y'} = \mathsf{Decode}(D_B, y')$ decapsulates successfully under $pk_{y'}$, since it knows the corresponding secret key.

If $\widetilde{Y}^*$ is the subset of $\widetilde{Y}$ whose associated ciphertexts decapsulate correctly, then Alice's protocol output will be $X \cap \widetilde{Y}^*$. Thus, Bob's behavior in the protocol can be simulated in the ideal world by choosing ideal input $\widetilde{Y}^*$.

Proof. **Corrupt Alice:** Figure 6 contains the formal details of a sequence of hybrids that prove that the real and ideal worlds are indistinguishable. The simulator follows the intuition described above, and is described in the final hybrid of the figure. Below we summarize the hybrids, especially differences between consecutive hybrids, and justify why consecutive hybrids are indistinguishable.

- *Hybrid 0:* The real protocol interaction. The adversary Adv produces protocol message D_A, then receives D_B computed according to Bob's honest protocol. Throughout the entire interaction, Adv has access to oracles $\mathcal{O}_A, \mathcal{O}_B$.
- *Hybrid 1:* Instead of an honest $\mathcal{O}_A$ oracle, the OKVS extractor plays the role of $\mathcal{O}_A$. When Adv generates D_A we run the extraction algorithm to identify a set $\widetilde{X}$. The interaction aborts if, for any $y \in Y \setminus \widetilde{X}$, the value $\mathsf{Decode}_A(D_A, y)$ was not sampled honestly by the simulator (through the Samp algorithm). The hybrids are indistinguishable (and, in particular, the abort happens with negligible probability) by a straightforward reduction to the extractability property of OKVS_A.
- *Hybrid 2:* Instead of sampling truly random elements of $\mathcal{P}_\lambda$, the OKVS extractor (via the Samp oracle) samples KEM public keys. The change is indistinguishable by a straightforward reduction to the pseudorandom public key property of KEM.

  Later, when computing Bob's protocol message, for each $\widetilde{pk}_y = \mathsf{Decode}_A(D_A, y)$, we generate a KEM ciphertext. For those $y \notin \widetilde{X}$, for convenience we generate these ciphertexts early and preemptively at the time $\widetilde{pk}_y$ was sampled (in Samp). This change has no effect on the interaction.
- *Hybrid 3:* Each time Samp generates public keys it also generates an encapsulation. Instead of generating it honestly, it chooses the ciphertext uniformly. The change is indistinguishable by a straightforward reduction to the ciphertext pseudorandomness property of KEM.
- *Hybrid 4:* In hybrid 3, D_B is generated by encoding an OKVS, where values corresponding to $y \in Y \cap \widetilde{X}$ are valid encapsulations and values corresponding to $y \in Y \setminus \widetilde{X}$ are sampled uniformly. From obliviousness, this is indistinguishable from simply excluding the keys in $Y \setminus \widetilde{X}$. Making this change gives us hybrid 4.
- *Ideal interaction:* Hybrid 4 only uses $Y \cap \widetilde{X}$, and thus it describes steps that can be carried out by the simulator in the ideal world. In summary, the simulator plays the role of OKVS extractor. After receiving D_A from the adversary, it extracts $\widetilde{X}$ and sends it to the $\mathcal{F}_{\text{psi}}$ functionality. It receives $Z = Y \cap \widetilde{X}$ in response, and uses this to generate a simulated OKVS D_B from Bob.

Corrupt Bob: Like the previous case, we describe the hybrids and simulator in detail in Figure 7 and only summarize them here.

- *Hybrid 0:* The real protocol interaction. The adversary receives protocol message D_A, generated according to Alice's honest protocol. Then the adversary sends a response D_B and Alice produces output according to the protocol. Throughout the entire interaction, Adv has access to oracles $\mathcal{O}_A, \mathcal{O}_B$.

Real interaction:
($\mathcal{O}_A, \mathcal{O}_B$ simulated honestly)

receive D_A from adversary

for $y \in Y$:
  $\widetilde{pk}_y = \mathsf{Decode}_A^{\mathcal{O}_A}(D_A, y)$
  $(C_y, -) = \mathsf{Encaps}(\widetilde{pk}_y)$
  $S_B = S_B \cup \{(y, C_y)\}$
$D_B = \mathsf{Encode}_B^{\mathcal{O}_B}(S_B)$

give D_B to adversary

Hybrid 1:
$\mathsf{ExtOracle}^{\mathsf{Samp}}$ plays role of $\mathcal{O}_A$
($\mathcal{O}_B$ simulated honestly)

$\mathsf{Samp}()$:
  $r \leftarrow \mathsf{KEM}.\mathcal{P}_\lambda$
  $\mathcal{Q} = \mathcal{Q} \cup \{r\}$
  return r

receive D_A from adversary
$\widetilde{X} = \mathsf{ExtOKVS}(D_A)$

for $y \in Y$:
  $\widetilde{pk}_y = \mathsf{Decode}_A^{\mathcal{O}_A}(D_A, y)$
  if $y \notin \widetilde{X}$:
    abort if $\widetilde{pk}_y \notin \mathcal{Q}$
  $(C_y, -) = \mathsf{Encaps}(\widetilde{pk}_y)$
  $S_B = S_B \cup \{(y, C_y)\}$
$D_B = \mathsf{Encode}_B^{\mathcal{O}_B}(S_B)$

give D_B to adversary

Hybrid 2:
$\mathsf{ExtOracle}^{\mathsf{Samp}}$ plays role of $\mathcal{O}_A$
($\mathcal{O}_B$ simulated honestly)

$\mathsf{Samp}()$:
  $(-, pk) \leftarrow \mathsf{KeyGen}(\lambda)$
  $(\widehat{C}_{pk}, -) = \mathsf{Encaps}(pk)$
  $\mathcal{Q} = \mathcal{Q} \cup \{pk\}$
  return pk

receive D_A from adversary
$\widetilde{X} = \mathsf{ExtOKVS}(D_A)$

for $y \in Y$:
  $\widetilde{pk}_y = \mathsf{Decode}_A^{\mathcal{O}_A}(D_A, y)$
  if $y \notin \widetilde{X}$:
    abort if $\widetilde{pk}_y \notin \mathcal{Q}$
    $C_y = \widehat{C}_{\widetilde{pk}_y}$
  else: $(C_y, -) = \mathsf{Encaps}(\widetilde{pk}_y)$
  $S_B = S_B \cup \{(y, C_y)\}$
$D_B = \mathsf{Encode}_B^{\mathcal{O}_B}(S_B)$

give D_B to adversary

Hybrid 3:
$\mathsf{ExtOracle}^{\mathsf{Samp}}$ plays role of $\mathcal{O}_A$
($\mathcal{O}_B$ simulated honestly)

$\mathsf{Samp}()$:
  $(-, pk) \leftarrow \mathsf{KeyGen}(\lambda)$
  $\widehat{C}_{pk} \leftarrow \mathsf{KEM}.\mathcal{C}_\lambda$
  $\mathcal{Q} = \mathcal{Q} \cup \{pk\}$
  return pk

receive D_A from adversary
$\widetilde{X} = \mathsf{ExtOKVS}(D_A)$

for $y \in Y$:
  $\widetilde{pk}_y = \mathsf{Decode}_A^{\mathcal{O}_A}(D_A, y)$
  if $y \notin \widetilde{X}$:
    abort if $\widetilde{pk}_y \notin \mathcal{Q}$
    $C_y = \widehat{C}_{\widetilde{pk}_y}$
  else: $(C_y, -) = \mathsf{Encaps}(\widetilde{pk}_y)$
  $S_B = S_B \cup \{(y, C_y)\}$
$D_B = \mathsf{Encode}_B^{\mathcal{O}_B}(S_B)$

give D_B to adversary

Hybrid 4:
$\mathsf{ExtOracle}^{\mathsf{Samp}}$ plays role of $\mathcal{O}_A$
($\mathcal{O}_B$ simulated honestly)

$\mathsf{Samp}()$:
  $(-, pk) \leftarrow \mathsf{KeyGen}(\lambda)$
  $\mathcal{Q} = \mathcal{Q} \cup \{pk\}$
  return pk

receive D_A from adversary
$\widetilde{X} = \mathsf{ExtOKVS}(D_A)$

for $y \in Y \cap \widetilde{X}$:
  $\widetilde{pk}_y = \mathsf{Decode}_A^{\mathcal{O}_A}(D_A, y)$
  $(C_y, -) = \mathsf{Encaps}(\widetilde{pk}_y)$
  $S_B = S_B \cup \{(y, C_y)\}$
$D_B = \mathsf{Encode}_B^{\mathcal{O}_B}(S_B)$

give D_B to adversary

Ideal interaction:
$\mathsf{ExtOracle}^{\mathsf{Samp}}$ plays role of $\mathcal{O}_A$
($\mathcal{O}_B$ simulated honestly)

$\mathsf{Samp}()$:
  $(-, pk) \leftarrow \mathsf{KeyGen}(\lambda)$
  $\mathcal{Q} = \mathcal{Q} \cup \{pk\}$
  return pk

receive D_A from adversary
$\widetilde{X} = \mathsf{ExtOKVS}(D_A)$

send $\widetilde{X}$ to $\mathcal{F}_{\text{psi}}$
receive $Z = \widetilde{X} \cap Y$ from $\mathcal{F}_{\text{psi}}$

for $y \in Z$:
  $\widetilde{pk}_y = \mathsf{Decode}_A^{\mathcal{O}_A}(D_A, y)$
  $(C_y, -) = \mathsf{Encaps}(\widetilde{pk}_y)$
  $S_B = S_B \cup \{(y, C_y)\}$
$D_B = \mathsf{Encode}_B^{\mathcal{O}_B}(S_B)$

give D_B to adversary

Figure 6: Hybrids for the case that Alice is corrupt. Y is honest Bob's input. Changes from the previous hybrid are highlighted.

Real interaction:
($\mathcal{O}_A, \mathcal{O}_B$ simulated honestly)

for $x \in X$:
  $(sk_x, pk_x) \leftarrow \mathsf{KeyGen}(\lambda)$
  $S_A = S_A \cup \{(x, pk_x)\}$
$D_A = \mathsf{Encode}_A^{\mathcal{O}_A}(S_A)$

give D_A to adversary
receive D_B from adversary

for $x \in X$:
  $\widetilde{C}_x = \mathsf{Decode}_B^{\mathcal{O}_B}(D_B, x)$
  if $\mathsf{Decaps}(sk_x, \widetilde{C}_x) \neq \perp$:
    $Z = Z \cup \{x\}$
output Z

Hybrid 1:
SimOracle plays role of $\mathcal{O}_A$
($\mathcal{O}_B$ simulated honestly)

for $x \in X$:
  $(sk_x, pk_x) \leftarrow \mathsf{KeyGen}(\lambda)$
  $S_A = S_A \cup \{(x, pk_x)\}$
  $\mathcal{V}[x] = pk_x$

KV(x):
  if $\mathcal{V}[x]$ undefined:
    $\mathcal{V}[x] \leftarrow \mathsf{KEM}.\mathcal{P}_\lambda$
  return $\mathcal{V}[x]$

$D_A = \mathsf{SimOKVS}^{\mathsf{KV}}(S_A)$

give D_A to adversary
receive D_B from adversary

for $x \in X$:
  $\widetilde{C}_x = \mathsf{Decode}_B^{\mathcal{O}_B}(D_B, x)$
  if $\mathsf{Decaps}(sk_x, \widetilde{C}_x) \neq \perp$:
    $Z = Z \cup \{x\}$
output Z

Hybrid 2:
SimOracle plays role of $\mathcal{O}_A$
($\mathcal{O}_B$ simulated honestly)

KV(x):
  if $\mathcal{V}[x]$ undefined:
    $(sk_x, pk_x) \leftarrow \mathsf{KeyGen}(\lambda)$
    $\mathcal{V}[x] = pk_x$
  return $\mathcal{V}[x]$

$D_A = \mathsf{SimOKVS}^{\mathsf{KV}}(\emptyset)$

give D_A to adversary
receive D_B from adversary

for $x \in X$:
  call KV(x) to ensure sk_x defined
  $\widetilde{C}_x = \mathsf{Decode}_B^{\mathcal{O}_B}(D_B, x)$
  if $\mathsf{Decaps}(sk_x, \widetilde{C}_x) \neq \perp$:
    $Z = Z \cup \{x\}$
output Z

Hybrid 3:
$\mathsf{ExtOracle}^{\mathsf{Samp}}$ plays role of $\mathcal{O}_A$
SimOracle plays role of $\mathcal{O}_B$

Samp():
  $C \leftarrow \mathsf{KEM}.\mathcal{C}_\lambda$
  $\mathcal{Q} = \mathcal{Q} \cup \{C\}$
  return r

KV(x):
  if $\mathcal{V}[x]$ undefined:
    $(sk_x, pk_x) \leftarrow \mathsf{KeyGen}(\lambda)$
    $\mathcal{V}[x] = pk_x$
  return $\mathcal{V}[x]$

$D_A = \mathsf{SimOKVS}^{\mathsf{KV}}(\emptyset)$

give D_A to adversary
receive D_B from adversary
$\widetilde{Y} = \mathsf{ExtOKVS}(D_B)$

for $x \in X$:
  call KV(x) to ensure sk_x defined
  $\widetilde{C}_x = \mathsf{Decode}_B^{\mathcal{O}_B}(D_B, x)$
  if $x \notin \widetilde{Y}$:
    abort if $\widetilde{C}_x \notin \mathcal{Q}$
  if $\mathsf{Decaps}(sk_x, \widetilde{C}_x) \neq \perp$:
    $Z = Z \cup \{x\}$
output Z

Hybrid 4:
$\mathsf{ExtOracle}^{\mathsf{Samp}}$ plays role of $\mathcal{O}_A$
SimOracle plays role of $\mathcal{O}_B$

Samp():
  $C \leftarrow \mathsf{KEM}.\mathcal{C}_\lambda$
  $\mathcal{Q} = \mathcal{Q} \cup \{C\}$
  return r

KV(x):
  if $\mathcal{V}[x]$ undefined:
    $(sk_x, pk_x) \leftarrow \mathsf{KeyGen}(\lambda)$
    $\mathcal{V}[x] = pk_x$
  return $\mathcal{V}[x]$

$D_A = \mathsf{SimOKVS}^{\mathsf{KV}}(\emptyset)$

give D_A to adversary
receive D_B from adversary
$\widetilde{Y} = \mathsf{ExtOKVS}(D_B)$

for $x \in X \cap \widetilde{Y}$:
  call KV(x) to ensure sk_x defined
  $\widetilde{C}_x = \mathsf{Decode}_B^{\mathcal{O}_B}(D_B, x)$
  if $\mathsf{Decaps}(sk_x, \widetilde{C}_x) \neq \perp$:
    $Z = Z \cup \{x\}$
output Z

Ideal interaction:
$\mathsf{ExtOracle}^{\mathsf{Samp}}$ plays role of $\mathcal{O}_A$
SimOracle plays role of $\mathcal{O}_B$

Samp():
  $C \leftarrow \mathsf{KEM}.\mathcal{C}_\lambda$
  $\mathcal{Q} = \mathcal{Q} \cup \{C\}$
  return r

KV(x):
  if $\mathcal{V}[x]$ undefined:
    $(sk_x, pk_x) \leftarrow \mathsf{KeyGen}(\lambda)$
    $\mathcal{V}[x] = pk_x$
  return $\mathcal{V}[x]$

$D_A = \mathsf{SimOKVS}^{\mathsf{KV}}(\emptyset)$

give D_A to adversary
receive D_B from adversary
$\widetilde{Y} = \mathsf{ExtOKVS}(D_B)$

for $y \in \widetilde{Y}$:
  call KV(y) to ensure sk_y defined
  $\widetilde{C}_y = \mathsf{Decode}_B^{\mathcal{O}_B}(D_B, y)$
  if $\mathsf{Decaps}(sk_y, \widetilde{C}_y) \neq \perp$:
    $\widetilde{Y}^* = \widetilde{Y}^* \cup \{y\}$

send $\widetilde{Y}^*$ to $\mathcal{F}_{\text{psi}}$
honest party outputs $X \cap \widetilde{Y}^*$

Figure 7: Hybrids for the case that Bob is corrupt. X is honest Alice's input. Changes from the previous hybrid are highlighted.

- *Hybrid 1:* The honest $\mathcal{O}_A$ oracle is replaced with the OKVS simulator. Likewise, honest Alice's OKVS D_A is replaced with a simulated OKVS. The OKVS simulator has access to an oracle KV (random everywhere except in positions indexed by Alice's set), and causes $\mathsf{Decode}(D_A, \cdot)$ to agree with $\mathsf{KV}(\cdot)$. The change is indistinguishable by a straightforward reduction to the simulatability property of OKVS_A.

- *Hybrid 2:* Instead of the KV oracle sampling its outputs uniformly, we modify it to sample outputs as KEM public keys (with secret key known to the simulator). The change is indistinguishable by a straightforward reduction to the pseudorandom public-key property of KEM. Since KV already output KEM public keys for values in Alice's set X, they are no longer a special case in KV after this change. Thus, $\mathsf{SimOKVS}$ can take $\emptyset$ as its argument.

- *Hybrid 3:* The honest $\mathcal{O}_B$ oracle is replaced with the OKVS extractor. When the adversary gives D_B, we run the OKVS extractor on it to obtain a set of keys $\widetilde{Y}$. Then, except with negligible probability, either $x \in \widetilde{Y}$ or $\mathsf{Decode}_B^{\mathcal{O}_B}(D_B, x)$ was chosen by the extractor (via Samp). The change is indistinguishable by a straightforward reduction to the extractability property of OKVS_B.

- *Hybrid 4:* Consider $x \in X \setminus \widetilde{Y}$ in Hybrid 3. With overwhelming probability, it must be that $\widetilde{C}_x = \mathsf{Decode}_B^{\mathcal{O}_B}(D_B, x)$ was chosen by Samp, as a uniform ciphertext in $\mathcal{C}_\lambda$. By the sparse ciphertext property of KEM, it is only with negligible probability that $\mathsf{Decaps}(sk_x, \widetilde{C}_x) \neq \perp$, causing x to be included in Alice's output Z. As such, it has negligible effect to simply exclude such values of x from Alice's output entirely. Making this change results in Hybrid 4.

- *Ideal interaction:* Taking stock of the final output Z in the previous hybrid, it contains item x only if $x \in X$, $x \in \widetilde{Y}$, and $\mathsf{Decaps}(sk_x, \widetilde{C}_x) \neq \perp$. The simulator can check the second and third of these conditions (it knows sk_x for all x). Define $\widetilde{Y}^*$ to be the subset of $\widetilde{Y}$ whose corresponding ciphertext decapsulates without error. Then Alice's output is simply $X \cap \widetilde{Y}^*$. Thus, the previous hybrid is identical to the ideal world in which the simulator sends $\widetilde{Y}^*$ to the ideal $\mathcal{F}_{\text{psi}}$ functionality.

□

5.1 Comparison to RT21 and CDJ16

Both our protocol and RT21 are descendants of the protocol of Choi, Dachman-Soled, Jarecki [11] ("CDJ16"). In CDJ16, parties run many instances of a private equality test (PEQT) protocol, one session for each input item. In PEQT, each party has a string as input, and the protocol outputs 1 if the strings are equal, and 0 otherwise. They embed the protocol messages of the PEQT into a polynomial and exchange this polynomial. These works pre-date the terminology of OKVS, but can easily be seen as using a polynomial as an OKVS.

RT21 can be viewed as a modification of CDJ16 with the following observations: (1) One can use a KEM in place of a PEQT. (2) Under suitable assumptions about the KEM (satisfied at least by Diffie-Hellman), one can re-use a KEM public key instead of using a separate key for each session.

Our protocol can be viewed as a variation on this theme, where we observe: (1) Even a KEM is somewhat overkill, and all that is needed is some kind of "liveness check", which a KEM can offer (the output key of the KEM is not needed); (2) If we use a separate KEM key for each session, then we can use modern PQ KEM candidates, and in fact we only need a very mild CPA-security assumption on the KEM.

Our protocol is naturally 2 rounds (2 message flows), whereas the "basic" version of RT21 requires 3. Only for Diffie-Hellman specifically can RT21 run in 2 messages.

A qualitative comparison of these protocols is given in Figure 8.

5.2 An Optimization

Looking ahead, the ideal permutation Π (inside our simulatable/extractable OKVS abstraction) is the overwhelming performance bottleneck in the protocol. In Appendix E we show that the ideal permutation can be removed from Bob's OKVS construction and the protocol remains secure (at the cost of a slightly worse

protocol	CDJ16 [11]	RT21 [39] (2-round variant)	ours
round 1	for each item: gen PEQT first message encode PEQT messages into OKVS send OKVS	for each item: gen KEM ciphertext encode ciphertexts into OKVS send OKVS	for each item: gen KEM public key encode public keys into OKVS send OKVS
round 2	for each item: gen PEQT response encode PEQT messages into OKVS send OKVS	generate **one** KEM public key send public key and KEM output keys	for each item: gen KEM ciphertext encode ciphertexts into OKVS send OKVS

Figure 8: Comparison of our protocol vs. RT21 and CDJ16. "PEQT" refers to private equality test protocol (each party has a string, the protocol reveals only whether the strings are equal). The 2-round variant of RT21 requires a KEM in which ciphertexts can be generated before learning the public key.

β parameter — about 3% worse for our ML-KEM instantiation). Please see the theorem statement there for precise terms and conditions: we require the KEM to satisfy a very slightly stronger variant of the *sparse ciphertexts* property (Definition 4), which is indeed satisfied by ML-KEM.

The intuition behind the optimization is as follows. First, the proof never requires simulatability of this OKVS — simple obliviousness is enough. Second, the extractability requirement is stronger than necessary. Of course, the simulator for a corrupt Bob must somehow extract his effective PSI input. We show that there is an alternative way to extract: In the security proof, Alice's OKVS D_A is already being simulated so that (for every k) $\mathsf{Decode}(D_A, k)$ is a KEM public key with secret key known to the simulator. Thus, the simulator can extract Bob's PSI input by trial decryption under all such KEM secret keys. This removes the need for the ideal permutation in our OKVS construction of Section 3.3, but we still need the random oracle to argue that the size of Bob's extracted set is bounded.

Unfortunately, this analysis requires treating the protocol as a monolith, so it does not enjoy the modularity of the original analysis. In particular, it crucially uses the fact that the OKVS values are KEM ciphertexts used in a certain way.

6 Implementation

We implemented our PSI protocol in C++, supporting different choices of underlying cryptographic primitives, particularly for the ideal permutation. The implementation supports both single-threaded and multi-threaded execution, as our protocol is highly parallelizable. The complete implementation is publicly available in https://github.com/osu-crypto/PQ-PSI.

6.1 Implementation Details

We use a security parameter $\kappa = 128$ and choose the step size s for Figure 4 as the largest divisor of $N - n$ satisfying $s \geq 2\lambda$ and $n - s \geq 2\lambda$.

For the OKVS component, we adopt an RB-OKVS [6] as our default choice, using the implementation from [26] with parameters specified in Figure 2. While one could consider using a polynomial-based OKVS for smaller input sizes (e.g., $n \leq 2^8$), this does not significantly improve performance, since the plain OKVS is not the main computational bottleneck in our protocol. We use SHA-256 as our random oracle.

For the ultra-wide permutation Π, we use our wide-block construction instantiated with different underlying designs, including Keccak-f[1600] [29] and SNEIK-f[512] [41] as building blocks. For Keccak-f[1600], we use the full 24 rounds as specified in [29] (denoted as Keccak-1600-24 in Figure 8). In addition, [5] suggests that 12 rounds provide a comfortable security margin, we also evaluate a reduced-round variant with 12 rounds (denoted as Keccak-1600-12 in Figure 8). We also evaluate an 8-round Feistel construction using SHAKE256 as the round function with domain separation. As an alternative wide-block construction, we also consider HCTR2 [13] with a fixed key. As mentioned in Appendix B, this is demonstrably *not* an ideal permutation, but it provides a lower-bound on the performance of our protocol.

Finally, for the key encapsulation mechanism, we consider both post-quantum and classical options. Our post-quantum instantiation uses ML-KEM-512 [30] together with Kemeleon [20]. Kemeleon converts the ML-KEM public keys and ciphertexts into pseudorandom fixed-length rows suitable for the OKVS/permutation

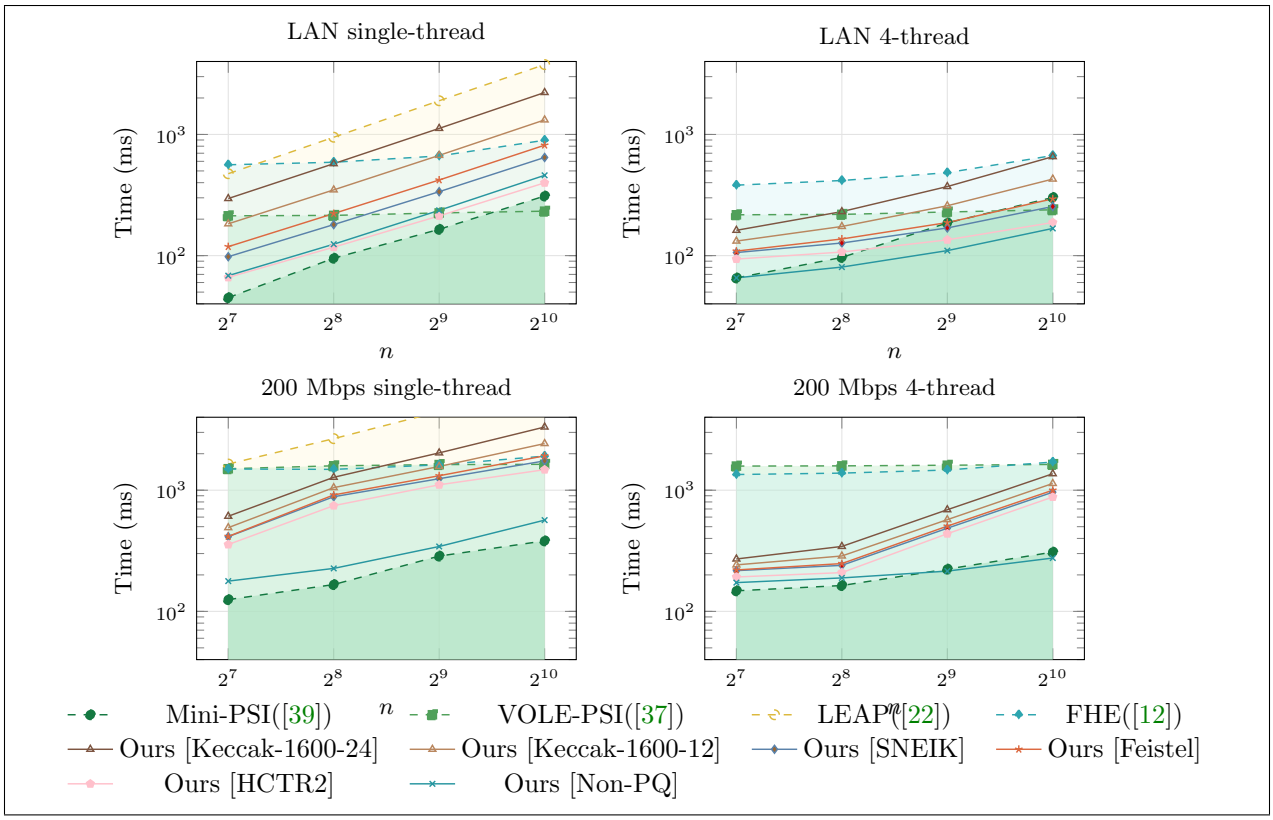

Figure 9: Scaling comparison under LAN and WAN settings. The green dashed lines represent Mini-PSI and VOLE-PSI, while the yellow and cyan dashed lines represent LEAP and FHE, respectively. The shaded regions indicate the area below the corresponding curves, where a protocol achieves lower running time than the corresponding scheme.

interface. For the non-post-quantum option ECDH+hash based on X25519 [25] and Elligator2 [4], this option is paired with Xoodoo 384-bit permutation [14].

Environment. All experiments were run on an Apple MacBook Pro with an Apple M1 Pro chip and 16 GB of RAM. To ensure a consistent software environment, benchmarks were all executed in Linux/amd64 Docker containers. All experiments were run on a single machine simulating two parties, where each party executes as a separate process. We evaluate both single-threaded and multi-threaded execution, using 4 threads per party. To simulate different deployment environments, we use the Linux `tc` command to configure network conditions. In particular, we consider two settings: a high-bandwidth LAN (10 Gbps) with a 0.2 ms round-trip delay, and a bandwidth-constrained setting of 200 Mbps with an 80 ms round-trip delay.

6.2 Performance Evaluation and Comparison

We evaluate the optimized version of our protocol from Section 5.2 against representative PSI protocols covering several different design paradigms. In particular, we compare against Mini-PSI [39], as a classical small-set PSI baseline; VOLE-PSI [37] instantiated with post-quantum base OTs, as a representative large-set OT/VOLE-based construction; LEAP [22], as a lattice-based OPRF approach to post-quantum PSI; and the lattice-based FHE-based PSI protocol [12]. We also evaluate an 8-round Feistel construction using SHAKE256 as the round function with domain separation, as an alternative approach to realizing the ultra-wide permutation required by our protocol. These protocols have substantially different cost structures and target different input-size regimes, so the comparison helps distinguish the cost of post-quantum security

itself from overhead specific to our KEM/OKVS-based small-set design.

All experiments are performed on the benchmark environment described above. We consider input sizes $n \in \{2^7, 2^8, 2^9, 2^{10}\}$, corresponding to the small-set regime targeted by our protocol. Table 1 reports total running time and communication, and Figure 9 illustrates the scaling behavior under the different network settings. We also provide a comparison between the optimized protocol (from Section 5.2) and the non-optimized version in Appendix F.

Mini-PSI [39]. Mini-PSI is the state-of-the-art maliciously secure PSI protocol for small input sets and serves as our primary non-post-quantum baseline. Like our protocol, Mini-PSI is specifically optimized for the regime where the parties hold only a few hundred items, making this comparison useful for measuring the overhead introduced by replacing classical Diffie–Hellman-style techniques with post-quantum primitives. We use the authors' publicly available implementation[6] and run it on the same benchmark machine and under the same network configurations as our implementation.

Compared with Mini-PSI, the performance of our protocol depends strongly on the choice of the wide-block permutation. Across all evaluated input sizes and network settings, the runtime ratios are 1.83×–8.70× for Keccak-1600-24, 1.39×–6.35× for Keccak-1600-12, 0.84×–5.31× for SNEIK, and 0.62×–4.48× for HCTR2. Thus, while the conservative Keccak-based instantiations are generally slower than Mini-PSI, the more efficient permutation choices can approach or even outperform it in some settings.

The communication difference is considerably larger: our PQ variants use approximately 26.7×–28.0× more communication than Mini-PSI over the evaluated input sizes. This gap should not be interpreted solely as an inherent cost of post-quantum PSI. On the computation side, ML-KEM itself has comparable performance to the classical KEM used in our non-PQ variant; most of the additional runtime comes from evaluating the ultra-wide permutation. On the communication side, the larger PQ public keys and ciphertexts are an important source of overhead, but our protocol also generates a separate KEM key pair for each input item, whereas Mini-PSI can exploit Diffie–Hellman-specific structure to reuse a key.

PQ VOLE-PSI [37]. VOLE-PSI represents a different design point intended primarily for large input sets. Such protocols pay a relatively large fixed setup cost for base OTs/VOLE, after which the per-item work consists mainly of inexpensive symmetric-key operations. This cost structure is highly effective for large sets because the setup can be amortized over many elements, but the fixed cost becomes significant in the small-set regime considered here.

We use the publicly available VOLE-PSI implementation[7] and modify only its base-OT layer to obtain post-quantum security. In particular, we use libOTe's built-in Masny–Rindal Kyber base-OT interface [27] with the pq-crystals Kyber768 reference backend[8]. This interface implements the Kyber-instantiated endemic OT of Masny and Rindal [27], which serves as the post-quantum base OT. These base OTs are then extended using the standard OT-extension to realize the OPRF required by VOLE-PSI. Since the OT-extension phase relies only on symmetric-key cryptographic operations, it preserves post-quantum security when instantiated with a post-quantum base OT. The higher-level VOLE-PSI protocol is left unchanged. Thus, this baseline represents the standard large-set VOLE-PSI design with its public-key setup replaced by a post-quantum alternative.

Our protocol is particularly competitive with PQ VOLE-PSI for small inputs. With HCTR2, it is faster in every evaluated setting except LAN single-thread at $n = 2^{10}$. With SNEIK, it is faster in LAN single-thread up to $n = 2^8$, in LAN 4-thread up to $n = 2^9$, in the 200 Mbps single-thread setting up to $n = 2^9$, and in the 200 Mbps 4-thread setting for all evaluated sizes. Keccak-1600-12 remains faster in LAN single-thread at $n = 2^7$, LAN 4-thread up to $n = 2^8$, 200 Mbps single-thread up to $n = 2^9$, and 200 Mbps 4-thread for all tested sizes. Even the conservative Keccak-1600-24 instantiation outperforms PQ VOLE-PSI in the 200 Mbps 4-thread setting for all tested input sizes.

[6] https://github.com/osu-crypto/MiniPSI
[7] https://github.com/Visa-Research/volepsi/
[8] https://github.com/pq-crystals/kyber

Communication exhibits a clear crossover. At $n = 2^7$, our protocol communicates 225 KiB compared with 1117.36 KiB for PQ VOLE-PSI, or only about 0.20× as much. At $n = 2^{10}$, however, our communication grows to 1712.50 KiB compared with 1163.03 KiB for VOLE-PSI, or about 1.47× as much. This behavior reflects the two protocols' different scaling strategies: VOLE-PSI pays a large nearly fixed setup cost and then has a low marginal cost per element, whereas the communication in our protocol grows approximately linearly with the number of input items. Consequently, our approach is better suited to small sets, while the VOLE-based design becomes more attractive as the set size increases.

LEAP [22]. LEAP represents a fundamentally different approach to post-quantum PSI. Rather than replacing the public-key component of an OT-based protocol, LEAP constructs a lattice-based OPRF and applies it to PSI. It therefore provides a useful comparison with a post-quantum design whose main public-key operation is a lattice-based OPRF rather than a KEM. We evaluate LEAP using the authors' publicly available implementation[9] in the same experimental environment and for the same input sizes as our protocol. The current LEAP implementation supports only single-threaded execution, so Table 1 reports single-thread results for this baseline.

In our experiments, even our most computationally expensive PQ instantiation, Keccak-1600-24, consistently outperforms LEAP across the evaluated small-set regime, by approximately 1.60×–1.70× in LAN and 2.08×–2.70× in the 200 Mbps setting. The performance gap becomes substantially larger with more efficient permutation instantiations: SNEIK is approximately 4.84×–5.84× faster in LAN and 3.01×–4.61× faster in the 200 Mbps setting, while HCTR2 is approximately 7.18×–9.46× and 3.57×–5.44× faster, respectively. LEAP also incurs substantially higher communication, requiring roughly 13× more data transfer across the tested input sizes. Overall, for the small-set regime considered here, our KEM/OKVS-based design provides a substantially more favorable computation–communication trade-off than the lattice-OPRF-based LEAP approach.

FHE-based PSI [12]. The FHE-based PSI protocol of Cong et al. represents another substantially different approach to post-quantum PSI. In this paradigm, one party encodes its input into homomorphic-encryption ciphertexts and the other party evaluates the intersection computation homomorphically. Compared with KEM-, OT-, or OPRF-based protocols, this approach requires substantially heavier public-key computation, but can process many elements together and amortize this cost over a large sender set. It is therefore primarily optimized for a different parameter regime from the small-set setting targeted by our work. We evaluate the FHE implementation on the same benchmark machine and network configurations and report both single-thread and 4-thread results in Table 1.

Using Keccak-1600-24 as a conservative baseline, in LAN single-thread, our protocol is approximately 1.03×–1.90× faster than FHE for $n \leq 2^8$, whereas FHE becomes approximately 1.69×–2.48× faster for $n \geq 2^9$. In WAN single-thread, our protocol is approximately 1.16×–2.45× faster for $n \leq 2^8$, while FHE is approximately 1.25×–1.74× faster for $n \geq 2^9$. Our more efficient permutation instantiations remain competitive over a broader range. Communication exhibits a similar crossover: our protocol uses approximately 1.73×–3.26× less communication for $n \leq 2^8$, whereas FHE uses approximately 1.13×–1.43× less for $n \geq 2^9$. Overall, the two approaches target complementary regimes, with our construction optimized for small sets and FHE-based PSI becoming increasingly attractive as the input size grows.

Feistel-based Domain Extension. As an alternative approach to realizing the required large-domain permutation, we implemented an 8-round Feistel construction using SHAKE256 as the round function. Feistel provides a natural way to construct a wide permutation from hash-based round functions and therefore serves as a useful comparison with our direct domain-extension construction. As shown in Table 1, Feistel is practical across all evaluated settings and performs substantially better than the conservative Keccak-1600-24 instantiation. In the LAN single-thread setting, its performance is also close to SNEIK and consistently better than Keccak-1600-12; for example, at $n = 2^7$, Feistel requires 118.61 ms, compared with 183.02 ms for Keccak-1600-12 and 97.99 ms for SNEIK. These results show that Feistel-based constructions provide a

[9] https://github.com/meyira/leap

practical alternative for instantiating the ultra-wide permutation required by our PSI protocol, while our construction differs in that it directly extends a smaller-domain ideal permutation such as Keccak-f.

Acknowledgments.

Ni Trieu was partially supported by NSF award #2451972, ARPA-H award #1AY2AX000167-01, and Amazon Research award.

References

[1] G. Alagic, E. Barker, L. Chen, D. Moody, A. Robinson, H. Silberg, and N. Waller. Recommendations for key-encapsulation mechanisms. Technical Report SP 800-227, NIST, 2023. https://csrc.nist.gov/pubs/sp/800/227/ipd.

[2] M. Bellare, V. T. Hoang, S. Keelveedhi, and P. Rogaway. Efficient garbling from a fixed-key blockcipher. In *2013 IEEE Symposium on Security and Privacy*, pages 478–492. IEEE Computer Society Press, May 2013.

[3] M. Bellare and P. Rogaway. On the construction of variable-input-length ciphers. In L. R. Knudsen, editor, *FSE'99*, volume 1636 of *LNCS*, pages 231–244. Springer, Berlin, Heidelberg, Mar. 1999.

[4] D. J. Bernstein, M. Hamburg, A. Krasnova, and T. Lange. Elligator: elliptic-curve points indistinguishable from uniform random strings. In A.-R. Sadeghi, V. D. Gligor, and M. Yung, editors, *ACM CCS 2013*, pages 967–980. ACM Press, Nov. 2013.

[5] G. Bertoni, J. Daemen, M. Peeters, G. Van Assche, R. Van Keer, and B. Viguier. KangarooTwelve: Fast hashing based on Keccak-p. In B. Preneel and F. Vercauteren, editors, *ACNS 2018*, volume 10892 of *LNCS*, pages 400–418. Springer, Cham, July 2018.

[6] A. Bienstock, S. Patel, J. Y. Seo, and K. Yeo. Near-optimal oblivious key-value stores for efficient PSI, PSU and volume-hiding multi-maps. In J. A. Calandrino and C. Troncoso, editors, *USENIX Security 2023*, pages 301–318. USENIX Association, Aug. 2023.

[7] R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd FOCS*, pages 136–145. IEEE Computer Society Press, Oct. 2001.

[8] D. Chakraborty, A. Dutta, and S. Kundu. Designing tweakable enciphering schemes using public permutations. Cryptology ePrint Archive, Report 2021/128, 2021.

[9] M. Chase and P. Miao. Private set intersection in the internet setting from lightweight oblivious PRF. In D. Micciancio and T. Ristenpart, editors, *CRYPTO 2020, Part III*, volume 12172 of *LNCS*, pages 34–63. Springer, Cham, Aug. 2020.

[10] S. Chen and J. Steinberger. Tight security bounds for key-alternating ciphers. Cryptology ePrint Archive, Paper 2013/222, 2013.

[11] C. Cho, D. Dachman-Soled, and S. Jarecki. Efficient concurrent covert computation of string equality and set intersection. In K. Sako, editor, *CT-RSA 2016*, volume 9610 of *LNCS*, pages 164–179. Springer, Cham, Feb. / Mar. 2016.

[12] K. Cong, R. C. Moreno, M. B. da Gama, W. Dai, I. Iliashenko, K. Laine, and M. Rosenberg. Labeled PSI from homomorphic encryption with reduced computation and communication. In Vigna and Shi [42], pages 1135–1150.

[13] P. Crowley, N. Huckleberry, and E. Biggers. Length-preserving encryption with HCTR2. Cryptology ePrint Archive, Report 2021/1441, 2021.

[14] J. Daemen, S. Hoffert, G. V. Assche, and R. V. Keer. The design of xoodoo and xoofff. *IACR Transactions on Symmetric Cryptology*, 2018(4):1–38, 2018.

[15] Y. Dai and J. P. Steinberger. Indifferentiability of 8-round Feistel networks. In M. Robshaw and J. Katz, editors, *CRYPTO 2016, Part I*, volume 9814 of *LNCS*, pages 95–120. Springer, Berlin, Heidelberg, Aug. 2016.

[16] D. Demmler, P. Rindal, M. Rosulek, and N. Trieu. PIR-PSI: Scaling private contact discovery. *PoPETs*, 2018(4):159–178, Oct. 2018.

[17] A. Dutta and M. Nandi. Tweakable HCTR: A BBB secure tweakable enciphering scheme. In D. Chakraborty and T. Iwata, editors, *INDOCRYPT 2018*, volume 11356 of *LNCS*, pages 47–69. Springer, Cham, Dec. 2018.

[18] E. Fujisaki and T. Okamoto. Secure integration of asymmetric and symmetric encryption schemes. In M. J. Wiener, editor, *CRYPTO'99*, volume 1666 of *LNCS*, pages 537–554. Springer, Berlin, Heidelberg, Aug. 1999.

[19] G. Garimella, B. Pinkas, M. Rosulek, N. Trieu, and A. Yanai. Oblivious key-value stores and amplification for private set intersection. In T. Malkin and C. Peikert, editors, *CRYPTO 2021, Part II*, volume 12826 of *LNCS*, pages 395–425, Virtual Event, Aug. 2021. Springer, Cham.

[20] F. Günther, D. Stebila, and S. Veitch. Obfuscated key exchange. In B. Luo, X. Liao, J. Xu, E. Kirda, and D. Lie, editors, *ACM CCS 2024*, pages 2385–2399. ACM Press, Oct. 2024.

[21] C. Guo, J. Katz, X. Wang, and Y. Yu. Efficient and secure multiparty computation from fixed-key block ciphers. In *2020 IEEE Symposium on Security and Privacy*, pages 825–841. IEEE Computer Society Press, May 2020.

[22] L. Heimberger, D. Kales, R. Lolato, O. Mir, S. Ramacher, and C. Rechberger. Leap: A fast, lattice-based OPRF with application to private set intersection. In S. Fehr and P.-A. Fouque, editors, *EUROCRYPT 2025, Part VII*, volume 15607 of *LNCS*, pages 254–283. Springer, Cham, May 2025.

[23] A. Heinrich, M. Hollick, T. Schneider, M. Stute, and C. Weinert. PrivateDrop: Practical privacy-preserving authentication for apple AirDrop. In M. Bailey and R. Greenstadt, editors, *USENIX Security 2021*, pages 3577–3594. USENIX Association, Aug. 2021.

[24] V. Kolesnikov, R. Kumaresan, M. Rosulek, and N. Trieu. Efficient batched oblivious PRF with applications to private set intersection. In E. R. Weippl, S. Katzenbeisser, C. Kruegel, A. C. Myers, and S. Halevi, editors, *ACM CCS 2016*, pages 818–829. ACM Press, Oct. 2016.

[25] A. Langley, M. Hamburg, and S. Turner. Elliptic Curves for Security. RFC 7748, Jan. 2016.

[26] J. Liu, P. Miao, M. Rosulek, X. Shi, and J. Wang. Updatable private set intersection from symmetric-key techniques. Cryptology ePrint Archive, Paper 2026/438, 2026.

[27] D. Masny and P. Rindal. Endemic oblivious transfer. In L. Cavallaro, J. Kinder, X. Wang, and J. Katz, editors, *ACM CCS 2019*, pages 309–326. ACM Press, Nov. 2019.

[28] U. M. Maurer, R. Renner, and C. Holenstein. Indifferentiability, impossibility results on reductions, and applications to the random oracle methodology. In M. Naor, editor, *TCC 2004*, volume 2951 of *LNCS*, pages 21–39. Springer, Berlin, Heidelberg, Feb. 2004.

[29] National Institute of Standards and Technology. Sha-3 standard: Permutation-based hash and extendable-output functions. `https://csrc.nist.gov/pubs/fips/202/final`, 2015. FIPS PUB 202.

[30] National Institute of Standards and Technology. Module-lattice-based key-encapsulation mechanism standard. Technical Report FIPS 203, U.S. Department of Commerce, 2024.

[31] NIST. Workshop on privacy-enhancing cryptography, 2024. https://csrc.nist.gov/Events/2024/wpec2024.

[32] J. Patarin. *The "Coefficients H" Technique*, page 328–345. Springer-Verlag, Berlin, Heidelberg, 2009.

[33] B. Pinkas, M. Rosulek, N. Trieu, and A. Yanai. SpOT-light: Lightweight private set intersection from sparse OT extension. In A. Boldyreva and D. Micciancio, editors, *CRYPTO 2019, Part III*, volume 11694 of *LNCS*, pages 401–431. Springer, Cham, Aug. 2019.

[34] B. Pinkas, M. Rosulek, N. Trieu, and A. Yanai. PSI from PaXoS: Fast, malicious private set intersection. In A. Canteaut and Y. Ishai, editors, *EUROCRYPT 2020, Part II*, volume 12106 of *LNCS*, pages 739–767. Springer, Cham, May 2020.

[35] B. Pinkas, T. Schneider, G. Segev, and M. Zohner. Phasing: Private set intersection using permutation-based hashing. In J. Jung and T. Holz, editors, *USENIX Security 2015*, pages 515–530. USENIX Association, Aug. 2015.

[36] B. Pinkas, T. Schneider, and M. Zohner. Faster private set intersection based on OT extension. In K. Fu and J. Jung, editors, *USENIX Security 2014*, pages 797–812. USENIX Association, Aug. 2014.

[37] P. Rindal and P. Schoppmann. VOLE-PSI: Fast OPRF and circuit-PSI from vector-OLE. In A. Canteaut and F.-X. Standaert, editors, *EUROCRYPT 2021, Part II*, volume 12697 of *LNCS*, pages 901–930. Springer, Cham, Oct. 2021.

[38] T. Ristenpart, H. Shacham, and T. Shrimpton. Careful with composition: Limitations of the indifferentiability framework. In K. G. Paterson, editor, *EUROCRYPT 2011*, volume 6632 of *LNCS*, pages 487–506. Springer, Berlin, Heidelberg, May 2011.

[39] M. Rosulek and N. Trieu. Compact and malicious private set intersection for small sets. In Vigna and Shi [42], pages 1166–1181.

[40] L. Roy, S. Lyakhov, Y. Jang, and M. Rosulek. Practical privacy-preserving authentication for SSH. In K. R. B. Butler and K. Thomas, editors, *USENIX Security 2022*, pages 3345–3362. USENIX Association, Aug. 2022.

[41] M.-J. O. Saarinen. SNEIK on microcontrollers: AVR, ARMv7-m, and RISC-v with custom instructions. Cryptology ePrint Archive, Paper 2019/936, 2019.

[42] G. Vigna and E. Shi, editors. *ACM CCS 2021*. ACM Press, Nov. 2021.

[43] P. Wang, D. Feng, and W. Wu. HCTR: a variable-input-length enciphering mode. In *International Conference on Information Security and Cryptology*, pages 175–188. Springer, 2005.

A Proof of Theorem 1

We prove the theorem in the following lemmas:

Lemma 1. *The construction in Theorem 1 is simulatable.*

Proof. Please refer to the hybrids shown on the left of Figure 10. Hybrid 0 is the interaction excluding the highlighted text. It is identical to the real interaction from the simulatability game for $\widetilde{\mathsf{Encode}}$.

Hybrid 1 is the interaction including the highlighted text. Hybrid 1 aborts (in lines $\star$ and $\star\star$) with negligible probability:

- In $\star$: $\mathsf{KV}(k)$ has just been uniformly chosen (since k does not appear in S), therefore it is negligibly likely that Π is already defined at $\mathsf{KV}(k)$.
- In $\star$: the OKVS D has just been generated. Since the OKVS is *linear* and k does not appear in S, we have that $z = \mathsf{Decode}(\mathsf{Encode}(S), k)$ is uniformly distributed with overwhelming probability.[10] Thus, for each k, it is negligibly likely that Π^{-1} is already defined at $z - H(k)$.
- In $\star\star$: $H(k)$ has just been sampled. Observe that this is also the first time that $\mathsf{KV}(k)$ can be called, so $\mathsf{KV}(k)$ is also freshly sampled. Thus, it is negligibly likely that Π is already defined at $\mathsf{KV}(k)$ or Π^{-1} is already defined at $z - H(k)$.

Conditioned on Hybrid 1 not aborting, it is indistinguishable from Hybrid 0. The only difference is whether $\Pi^{\pm}$ is programmed to give uniformly random outputs (as in Hybrid 1) or not programmed at all, and hence all its outputs are sampled without replacement (as in Hybrid 0). The difference is indistinguishable by a straight-forward birthday-bound argument.

As long as Hybrid 1 has not aborted, it is always true that $\Pi(\mathsf{KV}(k)) = \mathsf{Decode}(D, k) - H(k)$, by design. But this is equivalent to the condition $\widetilde{\mathsf{Decode}}(D, k) = \mathsf{KV}(k)$. Hence, if we replace "return 1" in the $\mathsf{Test}(k)$ oracle with "return $\widetilde{\mathsf{Decode}}(D, k) \stackrel{?}{=} \mathsf{KV}(k)$", we will see that Hybrid 1 is identical to the ideal game from the simulatability definition. The highlighted code describes the behavior or $\mathsf{SimOracle}^{\mathsf{KV}}$, and the lines defining S' and D describe the behavior of $\mathsf{SimOKVS}(S)$. □

Lemma 2. *The construction in Theorem 1 is β-extractable, for the β stated there.*

Proof. Please refer to the hybrids shown on the right of Figure 10. Hybrid 0 is the interaction excluding the highlighted text. It is identical to the real interaction from the extraction game for $\widetilde{\mathsf{Encode}}$.

Hybrid 1 is the interaction including the highlighted text. Hybrid 1 aborts only with negligible probability:

- In $\star$: the value $H(k)$ has just been sampled uniformly, so it is negligibly likely that $z - H(k)$ was a previous input/output of $\Pi^{\pm}$.
- In $\star\star$: the value $\Pi(v)$ has just been sampled almost-uniformly, so it is negligibly likely that it equals a previously-defined value of the form $\mathsf{Decode}(D, k) - H(k)$.

Suppose the adversary has made the oracle queries (to Π^{-1} and H) involved in the expression

$$v = \widetilde{\mathsf{Decode}}(D, k) = \Pi^{-1}\big(\mathsf{Decode}(D, k) - H(k)\big)$$

in Hybrid 1. Note that we have $\Pi(v) = \mathsf{Decode}(D, k) - H(k)$. There must have been some fresh query that established this point in the permutation — either a query to $\Pi(v)$ or to $\Pi^{-1}(\mathsf{Decode}(D, k) - H(k))$.

- If the query to $H(k)$ was made before the adversary provided D, then k was considered in the first highlighted for-loop. Either:
  - $k \in \tilde{K}$, or
  - $\mathsf{Decode}(D, k) - H(k)$ was *not* an output of a fresh Π query. This means the initial fresh query was $\Pi^{-1}\big(\mathsf{Decode}(D, k) - H(k)\big) \to v$.
- Otherwise, the query to $H(k)$ was made after D was provided. It must be the case that $\mathsf{Decode}(D, k) - H(k)$ cannot equal the output of a fresh Π query that happened *before* the $H(k)$ query — otherwise, Hybrid 1 would have aborted in $\star$. Also, $\mathsf{Decode}(D, k) - H(k)$ cannot equal the output of a fresh Π query that happened *after* the $H(k)$ query — otherwise, Hybrid 1 would have aborted in $\star\star$.

  We conclude that $\mathsf{Decode}(D, k) - H(k)$ was never the output of a fresh Π query, so the initial fresh query was
  $\Pi^{-1}\big(\mathsf{Decode}(D, k) - H(k)\big) \to v$.

[10] The $\mathsf{Probe}(k)$ vector is linearly independent of the constraints that define $\mathsf{Encode}(S)$, with overwhelming probability. Furthermore, Encode chooses a *random* solution to the linear constraints. This argument requires that the OKVS can support $|S| + 1$ items.

We conclude that either $k \in \tilde{K}$ or else the initial query $\Pi^{-1}(\mathsf{Decode}(D,k) - H(k)) \to v$ was fresh.

Suppose we make the following modifications to Hybrid 1:

- Instead of Π^{-1} oracle sampling its outputs to be permutation, it calls the Samp subroutine from the definition, which samples its outputs uniformly at random (i.e., with replacement). This change is indistinguishable by a standard permutation switching lemma argument. Now every output of a fresh Π^{-1}-query is contained in the set $\mathcal{Q}$, maintained by Samp.
- Replace "return 1" in Test with "return $k \in \tilde{K} \vee \widetilde{\mathsf{Decode}}(D,k) \in \mathcal{Q}$". From the reasoning above, the new expression is always true, so this change has no effect.

The result of these changes is identical to the ideal world in the extractability game. The highlighted text describes how $\mathsf{ExtOKVS}$ extracts from D to obtain $\tilde{K}$, and how $\mathsf{ExtOracle}$ simulates the H and $\Pi^{\pm}$ oracles (along with the fact that Π^{-1} simply calls Samp).

Now we must show that $|\tilde{K}| < \beta$, for the β stated in the theorem statement. We use a compression argument.

Claim 4. *If $\mathsf{Compress} : A \to B$ and $\mathsf{Uncompress} : B \to A$, then $\Pr_{a \leftarrow A}[\mathsf{Uncompress}(\mathsf{Compress}(a)) = a] \leq |B|/|A|$. In other words, the uniform distribution on A cannot be "compressed" into a smaller encoding space B.*

The claim follows from the fact that $\mathsf{Uncompress}$ has only $|B|$ possible outputs. Thus, $\mathsf{Uncompress}(\mathsf{Compress}(a)) = a$ is possible only if a happens to be one of these $|B|$ values. We apply this claim to argue that the random oracle H can be compressed in the following way.

Suppose $\mathcal{A}$ is an adversary in the extractability game making q queries to H, and let β be a bound on $|\tilde{K}|$. Define the following:

$\underline{\mathsf{Compress}}(r_1, \ldots, r_q, \Pi)$:
  run $\mathcal{A}$ as in the extractability game
    use r_i to answer the adversary's ith fresh query to H
    record k_i = the input to the ith query to H
    answer $\mathcal{A}$'s $\Pi^{\pm}$ queries with the provided oracle
    record s_j = result of jth fresh query to Π^{+}
  when $\mathcal{A}$ outputs D, the game computes $\widetilde{K}$
  abort if $|\widetilde{K}| < \beta$
  output (Π, D)
  $count = 1$
  for $i = 1$ to q:
    if $k_i \in \widetilde{K}$ and $count \leq \beta$:
⋆   output j such that $\mathsf{Decode}(D, k_i) - H(k_i) = s_j$
      $count := count + 1$
    else: output $H(k_i)$

It is important to note that the definition of $\tilde{K}$ ensures that the value j always exists in line $(\star)$ of $\mathsf{Compress}$.

As long as $\mathsf{Compress}$ doesn't abort (i.e., as long as $|\tilde{K}| \geq \beta$), its input can be recovered from its output

in the following way:

obtain (Π, D) from the input

run $\mathcal{A}$ as in the extractability game
  use Π to answer adversary's $\Pi^{\pm}$ queries
  record s_j = result of ith fresh query to Π^{+}
  when $\mathcal{A}$ makes the ith fresh query $H(k)$:
    if the next input token is an element of $\mathcal{V}$:
      use it for the output of the H query
    else, next input token is an index j:
      use $\mathsf{Decode}(D, k) - s_j$ as the output of the H query

The sequence of H-responses are the $r_1, \ldots, r_q$ values from the input to Compress.

Π and H have outputs from $\mathcal{V}$. Thus, the number of possible inputs to Compress is $|\mathcal{V}|^q \cdot |\mathcal{V}|!$. If D is a vector from $\mathcal{V}^\ell$, then the number of possible outputs of Compress is

$$\underbrace{|\mathcal{V}|!}_{\Pi} \cdot \underbrace{|\mathcal{V}|^\ell}_{D} \cdot \underbrace{\binom{q}{\beta} |\mathcal{V}|^{q-\beta} q^\beta}$$

The last term is because Compress outputs exactly β indices (from $[q]$) and exactly $q - \beta$ values from $\mathcal{V}$.

Thus, we have:

$$\begin{aligned}
\Pr[|\tilde{K}| \geq \beta] &= \Pr_a[\mathsf{Uncompress}(\mathsf{Compress}(a)) = a] \\
&\leq \frac{|\mathcal{V}|! \cdot |\mathcal{V}|^\ell \cdot \binom{q}{\beta} |\mathcal{V}|^{q-\beta} q^\beta}{|\mathcal{V}|^q \cdot |\mathcal{V}|!} \\
&= \binom{q}{\beta} |\mathcal{V}|^{\ell-\beta} q^\beta \\
&\leq |\mathcal{V}|^{\ell-\beta} q^{2\beta}
\end{aligned}$$

If we want to bound $\Pr[|\tilde{K}| \geq \beta] \leq 1/2^{-\kappa}$, then it suffices to choose β such that

$$\begin{aligned}
& |\mathcal{V}|^{\ell-\beta} q^{2\beta} \leq 1/2^{-\kappa} \\
\iff\ & (\ell - \beta) \log |\mathcal{V}| + 2\beta \log q \leq -\kappa \\
\iff\ & \beta(2 \log q - \log |\mathcal{V}|) \leq -\kappa - \ell \log |\mathcal{V}| \\
\iff\ & \beta \geq \frac{\kappa + \ell \log |\mathcal{V}|}{\log |\mathcal{V}| - 2 \log q}
\end{aligned}$$

where the direction of the inequality changes in the last step from dividing by a negative value. This is the value of β in the statement of the theorem, completing the proof. □

B Fixed-key HCTR is not an Ideal Permutation

The HCTR construction [43] is a strong pseudorandom permutation, summarized in Figure 11 and specified below. We describe a simplification that excludes HCTR's tweak feature.

Hybrid 0 / 1:
provide oracles:
$\Pi^{\pm}$: a random permutation on $\mathcal{V}$
H: a random oracle with range $\mathcal{V}$

receive S from adversary
KV = random oracle, except $\forall (k,v) \in S : \mathsf{KV}(k) = v$

// $\widetilde{\mathsf{Encode}}(S)$:
$S' = \left\{ \left(k, \Pi(v) + H(k)\right) \mid (k,v) \in S \right\}$
$D = \mathsf{Encode}(S')$
for every k already queried to H, where $\nexists v : (k,v) \in S$:
  $z = \mathsf{Decode}(D,k)$
$\star$ abort if $\Pi(\mathsf{KV}(k))$ or $\Pi^{-1}(z - H(k))$ already defined
  program $\Pi(\mathsf{KV}(k)) = z - H(k)$
give D to adversary

augment the behavior of the H oracle, as:

upon a *fresh* query $H(k)$:
  sample output $H(k) \leftarrow \mathcal{V}$ as usual
  $z = \mathsf{Decode}(D,k)$
$\star\star$abort if $\Pi(\mathsf{KV}(k))$ or $\Pi^{-1}(z - H(k))$ already defined
  program $\Pi(\mathsf{KV}(k)) = z - H(k)$

provide oracle:
$\mathsf{Test}(k)$:
  return 1

Hybrid 0 / 1:
provide oracles:
H: a random oracle with range $\mathcal{V}$
$\Pi^{\pm}$: a random permutation on $\mathcal{V}$

receive D from adversary
for every query k to the H oracle:
  $z = \mathsf{Decode}(D,k)$
  if $z - H(k)$ was the output of a *fresh* query to Π:
    $\widetilde{K} = \widetilde{K} \cup \{k\}$

augment the behavior of the oracles as:

upon *fresh* query $H(k)$:
  sample output $H(k) \leftarrow \mathcal{V}$ as usual
  $z = \mathsf{Decode}(D,k)$
$\star$ abort if $z - H(k)$ was output of prev.*fresh* query to Π

upon *fresh* query $\Pi(v)$:
  sample output $\Pi(v)$ as usual
$\star\star$abort if there exists a previous query $H(k)$
    ... such that $\mathsf{Decode}(D,k) - H(k) = \Pi(v)$

provide oracle:
$\mathsf{Test}(k)$:
  return 1

Figure 10: On the left: hybrids 0 (excluding highlighted text) and 1 (including highlighted text) for Lemma 1. On the right: hybrids for Lemma 2.

$\mathsf{HCTR}_{K_1,K_2,K_3,K_4}(X[1..N])$:
  $X_L = X[1..n]$
  $X_R = X[n+1..N]$
  $U = X_L \oplus H_{K_2}(X_R)$
  $V = E_{K_1}(U)$
  $Y_R = X_R \oplus CTR_{K_4}(U \oplus V)$
  $Y_L = H_{K_3}(Y_R) \oplus V$
  return $Y_L \| Y_R$

Here,

- E is a PRP with block length n;
- H is a keyed hash function with input length $N - n$ and output length n;
- CTR is the CTR-mode stream cipher, with output length $N - n$.
- HCTR has N-bit inputs and outputs.

Generally, n is close to the security parameter and N is much larger.

Claim 5. *HCTR with a fixed key* ***is not*** *indifferentiable from an ideal permutation, even if E, H, CTR are modeled as idealized primitives.*

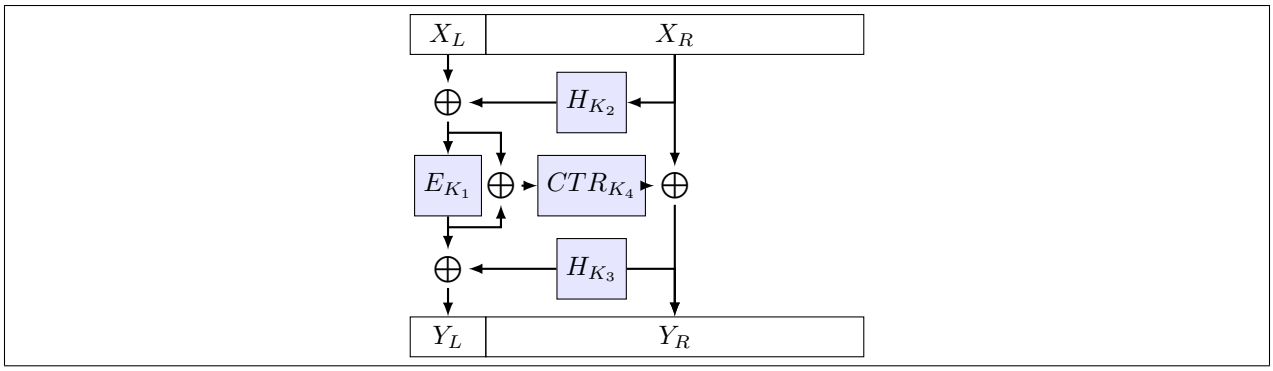

Figure 11: Overview of HCTR. E is a block cipher; H is a keyed hash function; CTR is the counter-mode key stream.

Proof. Choose k distinct values $X_R^{(1)}, \ldots, X_R^{(k)}$, arbitrarily. For each i, set $X_L^{(i)} = H_{K_2}(X_R^{(i)})$. Consider the computations $Y_L^{(i)} \| Y_R^{(i)} = \mathsf{HCTR}(X_L^{(i)} \| X_R^{(i)})$ for all i. By construction, each computation will have identical intermediate value U, and therefore V and therefore $CTR_{K_4}(U \oplus V)$. Thus, $X_R^{(i)} \oplus Y_R^{(i)}$ is identical for all i.

On the other hand, it is not hard to see that if Π is an ideal permutation and $N - n \gg \lambda$, it is infeasible to find $k \geq 2$ input-output pairs $Y^{(i)} = \Pi(X^{(i)})$ such that all $X^{(i)} \oplus Y^{(i)}$ agree in their final $N - n$ bits. □

HCTR is based on an earlier design of Bellare & Rogaway [3], and the previous claim applies to all descendants of this design, including modern HCTR variants [17] and the candidate NIST-standard HCTR2 [13].

C H-coefficient Method

We briefly recall the H-coefficient technique following Patarin [32] and Chen and Steinberger [10]: We consider an interaction between an adversary $\mathcal{A}$ and a game $\mathcal{G}$. Both are stateful, randomized programs. The adversary repeatedly sends a *query* to the game and receives a corresponding *response.* Finally, the adversary generates an output. Without loss of generality, we may assume that the output is a single bit, and that the adversary uses this output as its final query in the game.

Let $\mathcal{A} \circ \mathcal{G}$ denote the result of executing $\mathcal{A}$ and $\mathcal{G}$ together in the natural way. The output of $\mathcal{A} \circ \mathcal{G}$ is defined to be the *transcript,* consisting of the adversary's queries interleaved with the game's responses. Fixing an adversary $\mathcal{A}$ and game $\mathcal{G}$, the transcript $\mathsf{T} = (q_1, r_1, q_2, r_2, \ldots, q_n)$ arises with probability

$$\Pr_{\mathcal{A} \circ \mathcal{G}}[\mathsf{T}] = \Pr_{\mathcal{A} \circ \mathcal{G}}[q_1] \cdot \Pr_{\mathcal{A} \circ \mathcal{G}}[r_1 \mid q_1] \cdot \Pr_{\mathcal{A} \circ \mathcal{G}}[q_2 \mid q_1, r_1] \cdots \\ \cdots \Pr_{\mathcal{A} \circ \mathcal{G}}[q_n \mid q_1, r_1, \ldots, r_{n-1}].$$

Notably, the $\Pr[q_i]$ probabilities depend only on $\mathcal{A}$, while the $\Pr[r_i]$ probabilities depend only on $\mathcal{G}$. Thus, we may rearrange and write:

$$\Pr_{\mathcal{A} \circ \mathcal{G}}[\mathsf{T}] = \left(\prod_i \Pr_{\mathcal{A}}[q_i \mid q_1, r_1, \ldots, r_{i-1}] \right) \left(\prod_i \Pr_{\mathcal{G}}[r_i \mid q_1, r_1, \ldots, q_i] \right) \\ \stackrel{\text{def}}{=} P_{\mathcal{A}}(\mathsf{T}) \cdot P_{\mathcal{G}}(\mathsf{T})$$

where $P_{\mathcal{A}}(\mathsf{T})$ and $P_{\mathcal{G}}(\mathsf{T})$ are defined to be the two products, respectively.

Let $\mathcal{T}$ denote the set of all transcripts, and let $\mathcal{T}_1$ denote the set of all transcripts in which the adversary outputs 1 (recall that the adversary's output is included in the transcript as its final query).

Adversary $\mathcal{A}$'s *advantage* in distinguishing games $\mathcal{G}_1$ from $\mathcal{G}_2$ is defined to be:

$$\mathsf{Adv}_{\mathcal{A}}(\mathcal{G}_1, \mathcal{G}_2) \stackrel{\text{def}}{=} \Pr[\mathcal{A} \circ \mathcal{G}_1 \in \mathcal{T}_1] - \Pr[\mathcal{A} \circ \mathcal{G}_2 \in \mathcal{T}_1]$$

Lemma 3. *Let $\mathcal{G}_1$ and $\mathcal{G}_2$ be two games and let $\mathcal{A}$ be an adversary. Let $\mathsf{Bad} \subseteq \mathcal{T}$ be a subset of transcripts. Suppose that for all $\mathsf{T} \in \mathcal{T} \setminus \mathsf{Bad}$ we have $1 - \frac{P_{\mathcal{G}_1}(\mathsf{T})}{P_{\mathcal{G}_2}(\mathsf{T})} \leq \epsilon$. Then,*

$$\mathit{Adv}_{\mathcal{A}}(\mathcal{G}_1, \mathcal{G}_2) \leq \Pr[\mathcal{A} \circ \mathcal{G}_1 \in \mathsf{Bad}] + \epsilon$$

Proof.

$$\begin{aligned}
&\mathsf{Adv}_{\mathcal{A}}(\mathcal{G}_1, \mathcal{G}_2) \\
&\quad = \Pr[\mathcal{A} \circ \mathcal{G}_1 \in \mathcal{T}_1] - \Pr[\mathcal{A} \circ \mathcal{G}_2 \in \mathcal{T}_1] \\
&\quad = \sum_{\mathsf{T} \in \mathcal{T}_1} \left(\Pr_{\mathcal{A} \circ \mathcal{G}_1}[\mathsf{T}] - \Pr_{\mathcal{A} \circ \mathcal{G}_2}[\mathsf{T}] \right) \\
&\quad = \sum_{\mathsf{T} \in \mathcal{T}_1 \cap \mathsf{Bad}} \left(\Pr_{\mathcal{A} \circ \mathcal{G}_1}[\mathsf{T}] - \Pr_{\mathcal{A} \circ \mathcal{G}_2}[\mathsf{T}] \right) + \sum_{\mathsf{T} \in \mathcal{T}_1 \setminus \mathsf{Bad}} \left(\Pr_{\mathcal{A} \circ \mathcal{G}_1}[\mathsf{T}] - \Pr_{\mathcal{A} \circ \mathcal{G}_2}[\mathsf{T}] \right) \\
&\quad \leq \sum_{\mathsf{T} \in \mathcal{T}_1 \cap \mathsf{Bad}} \Pr_{\mathcal{A} \circ \mathcal{G}_1}[\mathsf{T}] + \sum_{\mathsf{T} \in \mathcal{T}_1 \setminus \mathsf{Bad}} \Big(P_{\mathcal{A}}(\mathsf{T}) P_{\mathcal{G}_1}(\mathsf{T}) - P_{\mathcal{A}}(\mathsf{T}) P_{\mathcal{G}_2}(\mathsf{T}) \Big) \\
&\quad \leq \Pr[\mathcal{A} \circ \mathcal{G}_1 \in \mathsf{Bad}] + \sum_{\mathsf{T} \in \mathcal{T}_1 \setminus \mathsf{Bad}} P_{\mathcal{A}}(\mathsf{T}) P_{\mathcal{G}_1}(\mathsf{T}) \Big(1 - \frac{P_{\mathcal{G}_2}(\mathsf{T})}{P_{\mathcal{G}_1}(\mathsf{T})} \Big) \\
&\quad \leq \Pr[\mathcal{A} \circ \mathcal{G}_1 \in \mathsf{Bad}] + \sum_{\mathsf{T} \in \mathcal{T}_1 \setminus \mathsf{Bad}} \Pr_{\mathcal{A} \circ \mathcal{G}_1}[\mathsf{T}] \cdot \epsilon \\
&\quad \leq \Pr[\mathcal{A} \circ \mathcal{G}_1 \in \mathsf{Bad}] + \epsilon
\end{aligned}$$

□

D Proof of Theorem 2

D.1 Transcript Definition

A *raw transcript* consists of the adversary's oracle queries together with the corresponding responses. Formally, a transcript is a sequence of query-answer tuples of the form

$$(\pi^{\pm}, x, y) \quad \text{or} \quad (\Pi^{\pm}, X, Y).$$

D.1.1 Graph Representation

Given a transcript T, we define an associated graph $G(\mathsf{T})$ whose nodes and edges are specified as follows. Vertices are triples (x, y, β), where $x, y \in \{0,1\}^n$, $\beta \in \{0,1\}$, and $\pi(x) = y$. A vertex is called *invisible* if $\beta = 0$ and *visible* if $\beta = 1$. For now, suppose that all nodes are visible by default, and later we will discuss how invisible nodes can appear. A (labeled) edge

$$(m, n, -) \xrightarrow{i} (x, y, -)$$

indicates that

$$n[s+1:n] \oplus x[1:n-s] = i$$

Equivalently, $x[1:n-s] = n[s+1:n] \oplus i$.

$G(\mathsf{T})$ is defined as a function of the transcript and it makes sense in both real/ideal games. it's similar to, but separate from the graph that the simulator maintains in the ideal game (but we show in Lemma 12 that these two graphs correspond).

Definition 10. *A path in the graph is **consistent** with the construction iff the following conditions hold:*

1. ***Consecutive edge labels:*** *The edge-labels on the path are consecutive integers from $\{1,\ldots,r-1\}$, in increasing order.*

2. ***Short constraint:*** *For every pair of adjacent nodes $(x,y,-) \xrightarrow{i} (u,w,-)$, we have*

$$y[s+1:n] \oplus u[1:n-s] = i.$$

   We call these two nodes short neighbors.

3. ***Long constraint:*** *For every pair of nodes separated by t edges $(x,y,-) \xrightarrow{i} \cdots \xrightarrow{i+t-1} (u,w,-)$, we have*

$$y[1:s] = u[n-s+1:n].$$

   We call these two nodes long neighbors. (Of course, this constraint is not relevant for paths shorter than t nodes.)

As shown in figure 12, the *short constraints*, shown in **grey**, correspond to the i-edges between adjacent node as in figure 4; the *long constraints*, shown in **red**, connect nodes that are t steps apart. They match the segments $y_i[1:s]$ and $x_{i+t}[n-s+1:n]$. Unlike the short constraints, the long constraints do not correspond to edges in the graph.

Internal states and path types: Any consistent path consisting of at least t consecutive nodes is a *completable path*, which determines two *internal states*: a left internal state and a right internal state. In figure 12, the *left internal state* (shown in blue) is formed by the entire string x_i of the first node, together with the last s bits of $x_{i+1},\ldots,x_{i+t-1}$. Symmetrically, the *right internal state* is defined by the entire second component of the last node, together with the first s bits of the second components of the preceding $t-1$ nodes. In Figure 12, these correspond to y_{i+t-1} and $y_i,\ldots,y_{i+t-2}$, respectively. Note that node $i+t$ here is for illustrating the long constraint, and is not shown as part of the left or right internal states in Figure 12.

Specifically, a consistent path is *visibly completable* if it contains t consecutive visible nodes. A path is called *full* if it begins with a $\xrightarrow{1}$ edge and ends with a $\xrightarrow{r-1}$ edge. In this case, it corresponds to a complete execution of the construction and it determines two *end internal states*, which correspond to the input and output of $\Pi^{\pm}$.

Because any t consecutive nodes uniquely determine the internal state of the construction, we have the following:

Corollary 6. *If P_1 and P_2 are consistent paths, and they share a segment of k consecutive nodes, then either P_1 and P_2 are colinear ($P_1 \cup P_2$ is also a consistent path, and so they are both consistent with a single execution of Π), or $k < t$.*

In other words, two paths corresponding to different executions of Π cannot overlap in a segment of length t.

D.2 Game Definition

Real Game. The real game $(Cons^{\pi^{\pm}}, \pi^{\pm})$ is as follows. The game internally maintains a graph. Since there are no edges added to the graph, the graph in the real games is just a set of tuples.

- $\pi(x)$: If there exists y such that $(x,y,-)$ is a node in the graph, then return y. Otherwise, sample y uniformly, subject to the constraint that there is no node in the graph of the form $(-,y,-)$. Add $(x,y,1)$ to the graph and return y.

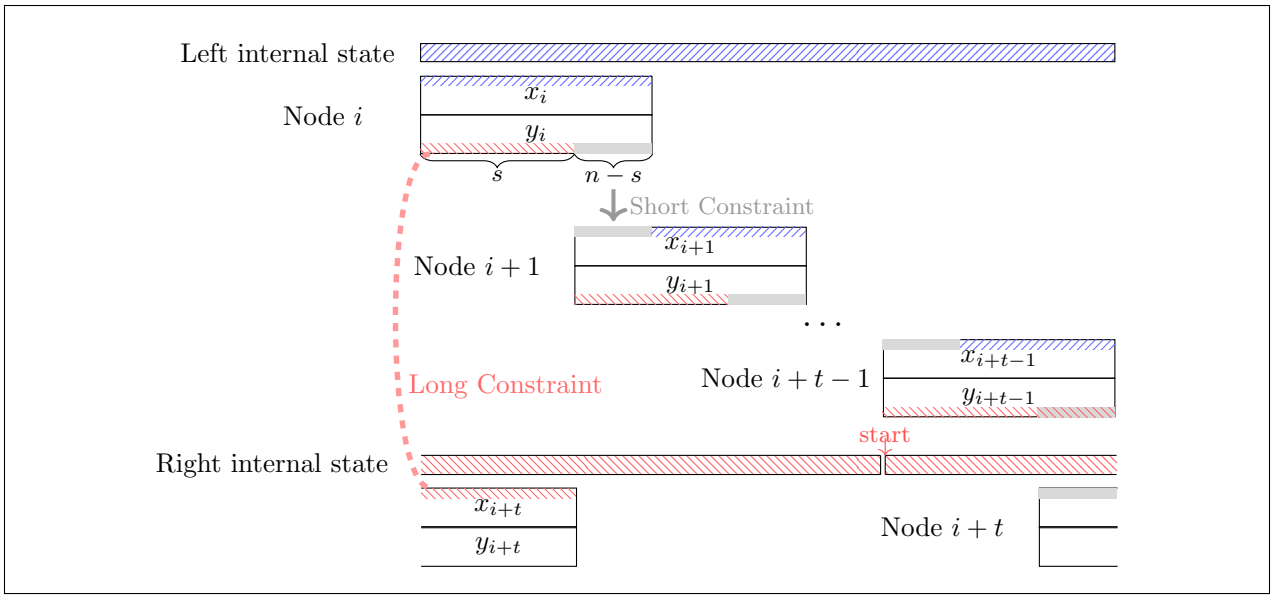

Figure 12: Illustration of internal states and short and long constraints.

- $\pi^{-1}(y)$: If there exists x such that $(x, y, -)$ is a node in the graph, then return x. Otherwise, sample x uniformly, subject to the constraint that there is no node in the graph of the form $(x, -, -)$. Add $(x, y, 1)$ to the graph and return x.
- $\mathsf{Cons}^{\pi^{\pm}}(X)$: run construction 3, where each call to π is processed using the above rules.
- $(\mathsf{Cons}^{-1})^{\pi^{\pm}}(Y)$: run the inverse of construction 3, where each call to π^{-1} is processed using the above rules.

Ideal Game. In the ideal world, the adversary $\mathcal{A}$ is given oracle access to a pair $(\Pi^{\pm}, \mathsf{Sim}^{\Pi^{\pm}})$, where $\Pi^{\pm}$ is an ideal permutation over $\{0,1\}^N$, and $\mathsf{Sim}^{\Pi^{\pm}}$ is a simulator with oracle access to $\Pi^{\pm}$.

Intuition behind our simulator. In the ideal game, the simulator maintains and keeps track of a graph G^*. The starting idea is to complete any path as soon as it becomes completable by adding π queries while ensuring consistency, also ensure the internal states at the left and right endpoints match with the input and output of Π. However, this approach fails because adding new nodes can make other paths completable, leading to infinite recursion and the simulator finish all completable paths in polynomial time. To fix this, a better idea is that the nodes added by simulator are marked as *invisible*. It only completes paths that are *visibly* completable, which causes no recursion and can ensure polynomial time as proved in lemma 8.

Another requirement is that every full path must match the corresponding $\Pi^{\pm}$. For a visibly completable path P, define its *closer endpoint* and *farther endpoint* as follows: Suppose the first visible node of P appears at position i along the full path P', if $i \le r/2 - t/2$, then the closer endpoint is the first node of the full path and the farther endpoint is the last node; otherwise, these roles are reversed.

The simulator first extends P to its closer endpoint. For instance, if the left end is closer, we recover the left internal state via $\mathsf{Reconstruct}(P, 0)$ and use it as input for Π. After obtaining the right-end internal state from Π, the simulator completes the farther side. To ensure consistency, this step requires t consecutive nodes that have not been sampled before. These are called *special nodes*, and we will provide more details on them in the following sections.

As illustrated in Figure 13, in the first case the closer endpoint is not visible. There are at most $t-1$ invisible nodes on each side of the visibly completable segment (as will be shown in Lemma 15), note that

a full path has $5t-2$ nodes, and the t nodes following these invisible nodes in the direction of the farther endpoint can serve as the special nodes. In the second case, the closer endpoint is visible, so the adversary can query Π to learn the other endpoint. This endpoint has at most $t-1$ visible nodes and an invisible span of length at most $t-1$, leaving at least t remaining nodes, from which we can select t nodes as the special nodes.

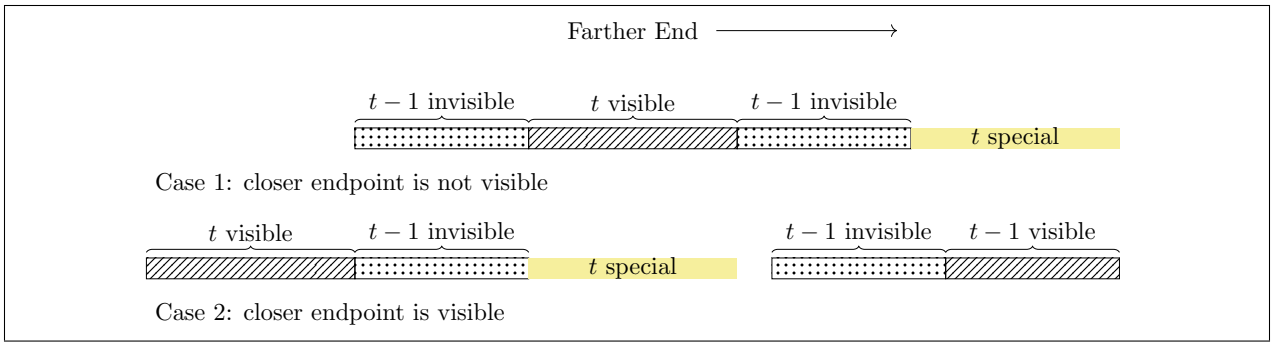

Figure 13: Intuition of the simulator. Suppose the left endpoint is the closer endpoint and the right endpoint is the farther endpoint. The figure shows the maximum number of visible and invisible nodes, and also highlights which nodes may become special nodes under the two cases.

Simulator details: The simulator details are in figure 14.

- $\pi(x), \pi^{-1}(y)$: It searches for a node of the form $(x, -, \beta)$ for $\pi(x)$(or $(-, y, \beta)$ for $\pi^{-1}(y)$) and creates it if necessary. Then call Complete when any visibly completable path appears. The $\pi^{\pm}$ call is made by the adversary, so the corresponding node is marked as visible.
- UpdateGraph(x, y, β): Update node $(x, y, -)$ in the graph and add all possible i-edges.
- Reconstruct(P, d): Reconstructs the left and right internal states of such a path, if $d = 0$, outputs the left internal states, if $d = 1$, outputs the right internal state.
- Complete(P): Let $P = (x_a, y_a, 0) \to \cdots \to (x_{i-1}, y_{i-1}, 0) \to (x_i, y_i, 1) \to \cdots \to (x_{i+t-1}, y_{i+t-1}, 1) \to (x_{i+t}, y_{i+t}, 0) \to \cdots \to (x_b, y_b, 0)$. (Edges are unlabeled for simplicity; the index i is omitted.) Nodes with indices $j \in [i,\, i+t-1]$ are visible, and all other nodes are invisible.

  The indices satisfy
  $$1 \le a \le i \le i+t-1 \le b \le r.$$
  The simulator extends the visibly completable path P to a full path in G^*. As illustrated by the intuition, the simulator first extends P to the closer endpoint, then extends it toward the farther endpoint along the longest possible invisible span. Finally, it reconstructs the internal state at the closer endpoint, calls Π to obtain the internal state at the farther endpoint, and completes the remaining nodes. The nodes sampled by the highlighted lines may contain special nodes, as shown in Figure 13. All new nodes added to the path are marked invisible, because the adversary has not queried them yet.

Extended Game. For the analysis, it is convenient to introduce an extended version of the normal game in both the real world and the ideal world. This extended game includes an additional oracle, denoted GoInvisible, which takes no input. Once GoInvisible is called, all future $\pi^{\pm}$-queries are marked as *invisible* (all $\pi^{\pm}$-queries before GoInvisible are visible). Here we are talking about things being marked invisible in the transcript, which is separate from whatever the simulator does in the ideal game. So whether the nodes in

$\underline{\pi(x)}$:
- if $\nexists y$ s.t. $(x, y, -)$ is a node in G^*: $y \xleftarrow{\$} \{0,1\}^n$
- $\mathsf{UpdateGraph}(x, y, 1)$
- $\forall$ P becomes visibly completable by $(x, y, 1)$:
  - $\mathsf{Complete}(P)$
- return y

$\underline{\pi^{-1}(y)}$:
- if $\nexists x$ s.t. $(x, y, -)$ is a node in G^*: $x \xleftarrow{\$} \{0,1\}^n$
- $\mathsf{UpdateGraph}(x, y, 1)$
- $\forall$ P becomes visibly completable by $(x, y, 1)$:
  - $\mathsf{Complete}(P)$
- return x

$\underline{\mathsf{UpdateGraph}(x, y, \beta)}$
- if $(x, y, -) \notin G^*$:
  - $G^* \leftarrow G^* \cup \{(x, y, \beta)\}$
  - if $\exists\, i \in [1, r-1]$, $(u, w, -) \in G^*$ s.t. $x[1 : n-s] = w[s+1 : n] \oplus i$:
    - for all such $(u, w, -)$:
      - $(u, w, -) \xrightarrow{i} (x, y, 0)$
  - if $\exists i \in [1, r-1]$ and $(u, w, -) \in G^*$ s.t. $y[s+1 : n] \oplus i = u[1 : n-s]$:
    - for all such $(u, w, -)$:
      - $(x, y, -) \xrightarrow{i} (u, w, -)$
- else:
  - if $\beta = 1$: update $(x, y, -)$ to $(x, y, 1)$ in G^*

$\underline{\mathrm{Reconstruct}(P, d)}$:
- $X \leftarrow 0^N$
- if $d == 0$:
- // left internal state
  - $X[1 : n] \leftarrow x_{a+t-1}$
  - for $j = a + t - 2$ down to a:
    - $X \leftarrow \mathrm{RotR}_s(X)$
    - $X[1 : n] \leftarrow x_j$
- if $d == 1$:
- // right internal state
  - $X[1 : n] \leftarrow y_{b-t+1}$
  - for $j = b - t + 2$ to b:
    - $X \leftarrow \mathrm{RotL}_s(X)$
    - $X[1 : n] \leftarrow y_j$
- return X

$\underline{\mathsf{Complete}(P)}$:
- // $P = (x_a, y_a, 0) \to \cdots \to (x_i, y_i, 1) \to \cdots$
- $\to (x_{i+t-1}, y_{i+t-1}, 1) \to \cdots \to (x_b, y_b, 0)$.
- // In this function, "sample" refers only to sampling nodes that
- // do not yet exist in G^*.
- if $i \le r/2 - t/2$: // left endpoint is closer
  - sample all (x_j, y_j) for $j \in [1, a-1]$ uniformly subj. to:
    - $y_j[s+1 : n] \oplus x_{j+1}[1 : n-s] = j$
    - $y_j[1 : s] = x_{j+t}[n-s+1 : n]$
  - sample all (x_j, y_j) for $j \in [b+1, i+2t-2]$ uniformly subj. to:
    - $y_{j-1}[s+1 : n] \oplus x_j[1 : n-s] = j$
    - $y_{j-t}[1 : s] = x_j[n-s+1 : n]$
  - $X \leftarrow \mathsf{Reconstruct}(P, 0)$
  - $Y \leftarrow \Pi(X)$
  - Sample all (x_j, y_j) for $j \in [i+2t-1, r]$ uniformly subj. to:
    - $j = r \Rightarrow y_j = Y[1 : n]$
    - $j < r \Rightarrow y_j[s+1 : n] \oplus x_{j+1}[1 : n-s] = j$
    - $y_j[1 : s] = \begin{cases} Y[N-(r-j)s+1 : N-(r-j-1)s], & j \ge r-t+1, \\ x_{j+t}[n-s+1 : n], & j < r-t+1 \end{cases}$
    - (x_{j-t}, y_{j-t}) exists $\Rightarrow y_{j-t}[1 : s] = x_j[n-s+1 : n]$

- if $i > r/2 - t/2$:// right endpoint is closer
  - sample all (x_j, y_j) for $j \in [b+1, r]$ uniformly subj. to:
    - $y_{j-1}[s+1 : n] \oplus x_j[1 : n-s] = j$
    - $y_{j-t}[1 : s] = x_j[n-s+1 : n]$
  - sample all (x_j, y_j) for $j \in [i-t+1, a-1]$ uniformly subj. to:
    - $y_j[s+1 : n] \oplus x_{j+1}[1 : n-s] = j$
    - $y_j[1 : s] = x_{j+t}[n-s+1 : n]$
  - $Y \leftarrow \mathsf{Reconstruct}(P, 1)$
  - $X \leftarrow \Pi^{-1}(Y)$
  - Sample all (x_j, y_j) for $j \in [1, i-t]$ uniformly subj. to:
    - $j = 1 \Rightarrow x_j = X[1 : n]$
    - $j > 1 \Rightarrow y_{j-1}[s+1 : n] \oplus x_j[1 : n-s] = j$
    - $x_j[1 : s] = \begin{cases} X[n+(j-2)s+1 : n+(j-1)s], & j \le t, \\ y_{j-t}[n-s+1 : n], & j > t \end{cases}$
    - (x_{j+t}, y_{j+t}) exists $\Rightarrow y_j[1 : s] = x_{j+t}[n-s+1 : n]$

- for all (x_j, y_j) newly sampled in this function:
  - $\mathsf{UpdateGraph}(x_j, y_j, 0)$

Figure 14: Simulator $\mathsf{Sim}^{\pi^{\pm}}$ in the ideal game. Operations related to special nodes are highlighted.

$G(\mathsf{T})$ are marked invisible or visible is based on whether GoInvisible was called previously. We only consider *normal adversaries*, namely adversaries that never invoke GoInvisible. Therefore, for any normal adversary, the extended game is identical to the original game.

Definition 11. *A transcript* T *(in the extended game) is* totally completed *if the following two conditions hold.*

1. *If P is a visibly completable path in $G(\mathsf{T})$ (visibly completable means all these nodes are marked visible in T), then there is also a* full *path (not necessarily visible) also in $G(\mathsf{T})$ that contains P.*
2. *If T contains a query $(\Pi^{\pm}, X, Y)$ then $G(\mathsf{T})$ contains a* full *path (not necessarily visible) whose left internal state is X.*

Lemma 4. *For every adversary $\mathcal{A}$ in the normal game, there exists an adversary $\mathcal{A}'$ in the extended game that always produces totally completed transcripts, while preserving the distribution of the transcript where only A is making queries. We will prove $\mathcal{A}'$ runs in polynomial time in lemma 8.*

Proof. Given an adversary A in the normal game, we construct an adversary A' in the extended game as follows:

- Run $\mathcal{A}$ until it terminates, giving output b. Forward $\mathcal{A}$'s oracle queries to the corresponding oracle in the extended game. Store the transcript T of $\mathcal{A}$'s oracle queries.
- Call $\mathsf{GoInvisible}$ to make all future $\pi^{\pm}$ queries marked invisible.
- For every query $(\Pi^{\pm}, X, Y) \in \mathsf{T}$ run the $\mathsf{Cons}^{\pi^{\pm}}(X)$ construction, making the necessary queries to the $\pi^{\pm}$ oracle (any new $\pi^{\pm}$ queries will be marked "invisible" in the extended transcript).
- For every visibly completable path P in $G(\mathsf{T})$, computes the internal states, make all π queries needed to extend this path to a full path, and also make the corresponding $\Pi^{\pm}$ query.
- Finally, output b.

Since all additional queries are invisible, they do not alter the visible transcript. Moreover, these invisible queries cannot create any new visibly completable paths. Therefore, the resulting transcript is totally completed, while the distribution of the visible transcript (and adversary's output) remains unchanged. □

By lemma 4, we will only consider totally completed transcripts in the following part.

Definition 12. *If T is a totally completed transcript, define $G(\mathsf{T}, \tau)$ to be the following graph:*

- *Let $\mathsf{T}_{\leq\tau}$ denote the transcript T restricted to the first τ queries. Construct the graph $G(\mathsf{T}_{\leq\tau})$ as usual.*
- *For every visibly completable path P in this graph, since T is totally completed, it contains a* full *path P' that contains P. Add all of P' to the graph, marking any new nodes as invisible.*

Lemma 5. *Let T be a totally completed transcript generated in the ideal game. Then, at time τ, the adversary has the τ-th query, the graph $G(\mathsf{T}, \tau)$ is exactly the graph G^* maintained by the simulator at that point.*

Proof. We compare the vertex sets of the two graphs.

By definition, the simulator's graph contains:

1. all visible $\pi^{\pm}$-queries issued up to time τ; and
2. for every visibly completable path present up to time τ, all nodes on its completed path.

This is exactly the definition of $G(\mathsf{T}, \tau)$.

It remains to justify that every invisible node added by the simulator is also represented in T. This follows from the assumption that T is totally completed: every visibly completable path is eventually completed in the transcript, possibly using invisible $\pi^{\pm}$-queries. Thus, every invisible node introduced by the simulator at time τ will eventually appear in T, and therefore is included in $G(\mathsf{T}, \tau)$ by definition. Thus the simulator's graph and $G(\mathsf{T}, \tau)$ coincide after the τ-th query. □

Definition 13. *Let P be a visibly completable path segment. Since the transcript is totally completed, let P' be the full path in the transcript that contains P. On P', define $\mathsf{SpecialNodes}(P)$ to be the set of t consecutive nodes $[i + 2t - 1 : i + 3t - 2]$ if the farther endpoint is the right endpoint, $[i - 2t + 1 : i - t]$ if the farther endpoint is the left endpoint. These nodes are sampled in a different way so that they can satisfy the long constraint.*

D.3 Indifferentiability Proof

Definition 14 (Bad totally completed transcripts)**.** *We define a set* $\mathsf{Bad} \subseteq \mathcal{T}$ *of bad extended transcripts:*

$$\mathsf{Bad} \;=\; \mathsf{Bad}_1 \cup \mathsf{Bad}_2 \cup \mathsf{Bad}_3 \cup \mathsf{Bad}_4,$$

where the events Bad_i *are specified below:*

- Bad_1 *(Ideal Permutation): The transcript is inconsistent with either* $\pi^{\pm}$ *or* $\Pi^{\pm}$ *being a permutation. That is, the transcript contains* (π, x, y) *and* (π, x, y') *for some* $y \neq y'$*, or it contains* (π, x, y) *and* (π, x', y) *for some* $x \neq x'$*. The same condition applies to* Π*.*
- Bad_2 *(*Π *Consistency):* $\pi^{\pm}$ *is inconsistent with* $\Pi^{\pm}$*, meaning that* T *contains* (Π, X, Y)*, but also contains a collection of* π*-queries whose answers determine* $\mathsf{Cons}^{\pi}(X) \neq Y$*, or determine* $\mathsf{Cons}^{\pi}(X') = Y$ *for* $X' \neq X$*.*
- Bad_3 *(Non-Forest): For some* τ*,* $G(\mathsf{T}, \tau)$ *is not a forest. This bad event will be useful in lemma 8.*
- Bad_4 *(Collision): The adversary's* τ*'th query results in a new visible completable path* P*; however, some of the nodes in* $\mathsf{SpecialNodes}(P)$ *already exist in* $G(\mathsf{T}, \tau - 1)$*.*

Lemma 6. *Let* $\mathsf{T} \notin \mathsf{Bad}$*, and let* q *be the number of fresh* $\pi^{\pm}$*-queries in* T*. Then*

$$P_{\mathcal{G}_{\text{real}}}(\mathsf{T}) = \frac{1}{(2^n)_q}.$$

Proof. In the real world, the transcript is generated by lazy sampling a random permutation π. Thus, the only source of randomness comes from answering fresh $\pi^{\pm}$-queries. Since $\mathsf{T} \notin \mathsf{Bad}_1$, the transcript is consistent with $\pi^{\pm}$ being a permutation, and no previously defined input or output is reused. Also because $\mathsf{T} \notin \mathsf{Bad}_2$, all the $\pi^{\pm}$ queries are consistent with their corresponding $\Pi^{\pm}$ queries. (Note that if $\mathsf{T} \in \mathsf{Bad}_2$ then $P_{\mathcal{G}_{\text{real}}}(\mathsf{T}) = 0$.) Therefore, the probability that all q fresh $\pi^{\pm}$-queries match T is

$$\frac{1}{2^n} \cdot \frac{1}{2^n - 1} \cdot \frac{1}{2^n - 2} \cdots \frac{1}{2^n - (q-1)} = \frac{1}{(2^n)_q}.$$

□

Lemma 7. *Let* $\mathsf{T} \notin \mathsf{Bad}$*. Let* q *be the number of fresh* $\pi^{\pm}$*-queries in* T*, and let* Q *be the number of* $\Pi^{\pm}$ *queries in* T*. Then*

$$P_{\mathcal{G}_{\text{ideal}}}(\mathsf{T}) = \frac{2^{NQ}}{(2^N)_Q \cdot 2^{nq}}.$$

Proof. In the ideal game, each $\Pi^{\pm}$-query is answered by sampling a random permutation over $\{0,1\}^N$, contributing a total factor of $1/(2^N)_Q$ to $\mathcal{P}_{\mathcal{G}_{\text{ideal}}}(\mathsf{T})$.

In the ideal game, nodes except $\mathsf{SpecialNodes}(P)$ for any P are sampled with answering $\pi^{\pm}$-queries randomly, contributing $1/2^n$ each. So their total contribution would be 2^{-nq}.

However, special nodes are sampled in a different way. For every Π-query, there is a corresponding completed path in the totally completed transcript. Such a path may be created either when the adversary makes a Π-query, or when the simulator detects a completable path and extends it. In either case, exactly t nodes on that path cannot be sampled uniformly, which are $\mathsf{SpecialNodes}(P)$. Since $\mathsf{T} \notin \mathsf{Bad}_4$, these t special nodes are indeed sampled by $\mathsf{Complete}(P)$ of the simulator.

Among these t queries: $t-1$ are sampled conditioned on partial information (either $x_{j+t}[n-s+1:n]$ or $y_{j-t}[1:s]$), each contributing a factor of $1/2^{n-s}$; the remaining node is used to connect two path segments, and is completely determined by the other nodes in the path (so it contributes no randomness). Thus, instead of contributing 2^{-nt} probability, these t queries contribute only $2^{-(n-s)(t-1)}$.

Therefore, each completed path increases the probability by a factor

$$\frac{2^{-(n-s)(t-1)}}{2^{-nt}} = 2^{nt-(n-s)(t-1)} = 2^N,$$

using $N = n + s(t-1)$. Since there are Q such completed paths, the total correction factor is 2^{NQ}, and we obtain

$$P_{\mathcal{G}_{\text{ideal}}}(\mathsf{T}) = \frac{2^{NQ}}{(2^N)_Q \cdot 2^{nq}}.$$

□

Lemma 8. *If* $\mathsf{T} \notin \mathsf{Bad}$, *Sim runs in polynomial time for each query, and* $\mathcal{A}'$ *runs in polynomial time in lemma 4.*

Proof. In the ideal game, the only extra work performed by Sim is to complete visibly completable paths in its graph. Since $\mathsf{T} \notin \mathsf{Bad}$, the simulator's graph is a forest. Moreover, each visible $\pi^{\pm}$-query adds at most one new visible node. So after τ adversarial queries, there are at most τ visible nodes, and therefore at most τ^2 different paths. Because of corollary 6, any two different visibly completable paths share at most $t-1$ nodes. Furthermore, since every visibly completable path has a length of at least t, it follows that there are at most a polynomial number of such paths. For each such path, the extension takes polynomial time. Moreover, all queries issued during completion are invisible and therefore do not create any new visibly completable paths. Thus, each query by the adversary triggers at most polynomially many completion steps, each of polynomial cost. Therefore, Sim runs in polynomial time per query.

And for the adversary A' in the extend game, for every visibly completable path in the original transcript, it runs the construction both forwards and backwards from each such state, and issues the corresponding $\Pi^{\pm}$ queries (which we have shown can be done in polynomial time). Furthermore, for every query to the $\mathsf{Cons}^{\pi^{\pm}}$ or $\Pi^{\pm}$ oracles, A' runs $\mathsf{Cons}^{\pi^{\pm}}$ in his head, and make the $\pi^{\pm}$ queries according to $\mathsf{Cons}^{\pi^{\pm}}$. Therefore, if $\mathcal{A}$ makes q $\pi^{\pm}$ queries and Q $\Pi^{\pm}$ queries, then $\mathcal{A}'$ makes $q + rQ$ $\pi^{\pm}$ queries and $Q + q^2/r$ $\Pi^{\pm}$ queries and the total number of queries is polynomial. So the overall process remains polynomial-time and A' runs in polynomial time as well.

□

Lemma 9. *In the ideal game, the probability that the transcript is bad is bounded by* $O(q^2r^2)/2^{n-s}$.

Proof. We prove it by lemmas below.

Lemma 10. $\Pr_{\mathsf{ideal}}[\mathsf{Bad}_1] \le O(\frac{q^2}{2^{n-s}})$.

Proof. For $\Pi^{\pm}$, the outputs are chosen to satisfy an ideal permutation, so the probability of an inconsistency in $\Pi^{\pm}$ is zero. For $\pi^{\pm}$, the simulator only samples the results in $\pi^{\pm}$ and Complete of figure 14, so that at least $n-s$ bits are uniformly random. The probability of the bad event triggered is at most $O(\frac{q}{2^{n-s}})$ for a single query. Since there are q $\pi^{\pm}$ queries in the ideal world, we have $\Pr_{\mathsf{ideal}}[\mathsf{Bad}_1] \le O(\frac{q^2}{2^{n-s}})$. □

Lemma 11. $\Pr_{\mathsf{ideal}}[\mathsf{Bad}_2] = 0$.

Proof. In the ideal world, a completed path can only be added to the graph by the simulator. For any $\Pi^{\pm}$, the simulator will either abort, or design the completed path to be consistent with Π as in figure 14. So $\Pr_{\mathsf{ideal}}[\mathsf{Bad}_2] = 0$. □

Lemma 12. *Define* $\mathrm{suffix}(x)$ *as the last* $n-s$ *bits of* x, *and* $\mathrm{prefix}(x)$ *as the first* $n-s$ *bits of* x. *For every node* $(x, y, -)$ *that exists in the graph* $G(\mathsf{T}, \tau)$ *at time* τ, *let* Pre_τ *denote the set of* $x[1 : n-s] \oplus i \oplus i'$ *for all* $i, i' \in [0, r-1]$ *at time* τ. *Similarly, let* Suf_τ *denote the set of* $y[s+1 : n] \oplus i \oplus i'$ *for all* $i, i' \in [0, r-1]$ *at time* τ. *Define the bad sub-event* $\mathsf{Bad}_{3.1}$*:*

- *After calling* $\pi(x) = y$ *at time* τ, $\mathrm{suffix}(y) \in \mathsf{Pre}_{\tau-1}$.

- *After calling* $\pi^{-1}(y) = x$ *at time* τ, $\text{prefix}(x) \in \mathsf{Suf}_{\tau-1}$.

$\Pr_{\mathsf{ideal}}[\mathsf{Bad}_{3.1}] \leq O(\frac{r^2 2^s q^2}{2^n})$.

Proof. For both special nodes and non-special nodes, the output of π is chosen such that the last $n-s$ bits are uniform (and for π^{-1}, the first $n-s$ bits are uniform). At time τ, the size of Pre, Suf is at most $O(qr^2)$, so a call to $\pi^{-1}(y) = x$ or $\pi(x') = y'$ has at most $O(\frac{r^2 2^s q}{2^n})$ probability to have $\text{prefix}(x) \in \mathsf{Suf}_{\tau-1}$ or $\text{suffix}(y') \in \mathsf{Pre}_{\tau-1}$. There are q calls of π^{-1} or π in the ideal world, so $\Pr_{\mathsf{ideal}}[\mathsf{Bad}_{3.1}] \leq O(\frac{r^2 2^s q^2}{2^n})$. □

Lemma 13. $\Pr_{\mathsf{ideal}}[\mathsf{Bad}_3] \leq \Pr_{\mathsf{ideal}}[\mathsf{Bad}_{3.1}]$.

Proof. Consider any node $(x, y, -)$, we bound the probability that this node closed a cycle or parallel paths in the graph at the time it was created.

Cycle: According to lemma 12: The output of $\pi(x)$ or $\pi^{-1}(y)$ is chosen so that unexpected i-edge to existing nodes are created with probability bounded by $\Pr_{\mathsf{ideal}}[\mathsf{Bad}_{3.1}]$. A cycle is formed when the output of $\pi^{\pm}$ connects to an existing node via an i-edge. Hence

$$Pr[\exists \tau,\, G(T, \tau) \text{ has cycles}] \leq \Pr_{\mathsf{ideal}}[\mathsf{Bad}_{3.1}].$$

Parallel paths: The following situations also happen with probability bounded by $\Pr_{\mathsf{ideal}}[\mathsf{Bad}_{3.1}]$:

- For $\pi(x)$, choose result y and $\exists(-, y', -)$ and $i, i' \in [1, r-1]$, $y'[s+1:n] \oplus i' = y[s+1:n] \oplus i$.
- For $\pi^{-1}(y)$, choose result x and $\exists(x', -, -)$ and $i, i' \in [1, r-1]$, $x'[1:n-s] \oplus i' = x[1:n-s] \oplus i$.

These situations mean that when $\pi(x)$ returns a new node $(x, y, -)$, there may exist another node $(x', y', -)$ in the graph such that $(x, y, -)$ and $(x', y', -)$ share a common neighbor.

Therefore, two distinct directed paths with the same start and end vertices (i.e., parallel paths) exists with probability $\leq \Pr_{\mathsf{ideal}}[\mathsf{Bad}_{3.1}]$. Consequently, the probability of the bad event that $\exists \tau$, $G(\mathsf{T}, \tau)$ is not a forest is bounded by $\Pr_{\mathsf{ideal}}[\mathsf{Bad}_3] \leq \Pr_{\mathsf{ideal}}[\mathsf{Bad}_{3.1}]$. □

Lemma 14. *(Neighbor-Visible Condition) Define the bad sub-event* $\mathsf{Bad}_{4.1}$*: the adversary makes a query* $(\pi^{\pm}, x, y)$ *at time* τ*, and in* $G(\mathsf{T}, \tau - 1)$*, invisible node* $(x, y, 0)$ *exists and for any path containing* $(x, y, 0)$*, either*

- $(x, y, 0)$ *is not an endpoint node, and its left and right neighbors are invisible, or*
- v *is an endpoint of the path, and the adversary has not queried* $\Pi^{\pm}$ *on inputs corresponding to this path.*

$\Pr_{\mathsf{ideal}}[\mathsf{Bad}_{4.1}] \leq O(\frac{q}{2^{n-s-1}})$.

Proof. A node is considered invisible if it has not been queried by the adversary. In this case, any sampling associated with the node is performed by the simulator and remains independent of the adversary's view. Therefore, from the adversary's perspective, this is equivalent to the node not having been sampled at all.

For the first situation, both the first $n-s$ bits of x and the last $n-s$ bits of y is unknown by the adversary at time τ, the probability of the adversary querying this node(either $\pi(x)$ or $\pi^{-1}(y)$) at time τ is at most $O(\frac{1}{2^{n-s-1}})$. For the second situation, either x of y has n unknown bits to the adversary, and the other one has at least $n-s$ unknown bits. So probability of the adversary querying this node at time τ is at most $O(\frac{1}{2^{n-s}} + \frac{1}{2^n})$. There are q $\pi^{\pm}$ queries in the ideal world, so $\Pr_{\mathsf{ideal}}[\mathsf{Bad}_{4.1}] \leq O(\frac{q}{2^{n-s-1}})$. □

Lemma 15. *Define bad subevent* $\mathsf{Bad}_{4.2}$ *as follows:* P *is a consistent path in* $G(\mathsf{T}, \tau)$ *that is not part of any full path, yet* P *contains a span of at least* t *consecutive invisible nodes. Then* $\Pr_{\mathsf{ideal}}[\mathsf{Bad}_{4.2}] \leq \Pr_{\mathsf{ideal}}[\mathsf{Bad}_{3.1}] + \Pr_{\mathsf{ideal}}[\mathsf{Bad}_{4.1}]$.

Proof. It suffices to show that for any invisible node v on P, the event that v is at distance at least $t-1$ from every visible node on P occurs only if either $\mathsf{Bad}_{3.1}$ or $\mathsf{Bad}_{4.1}$ happens. Let v be an invisible node on P. Invisible nodes are created only when the simulator extends a visibly completable path. Since P is not part of any full path, the node v must have been introduced while the simulator was completing some *different* path P'. Let P' denote this fully completed path.

We now argue that, except with probability at most $\Pr_{\mathsf{ideal}}[\mathsf{Bad}_{3.1}] + \Pr_{\mathsf{ideal}}[\mathsf{Bad}_{4.1}]$, the overlap between P and P' contains either a visible node, or the first invisible node adjacent to a visible node.

Indeed, suppose this is not the case. Then one of the following must happen. First, the simulator connects P' to an invisible node on P that is not adjacent to any visible node. By lemma 12, this occurs with probability at most $\Pr_{\mathsf{ideal}}[\mathsf{Bad}_{3.1}]$; Second, after the first invisible node adjacent to a visible node on P, the adversary queries a visible node that is adjacent to an invisible node (this node cannot be an endpoint if we want to make the span longer), thereby creating P'. By Lemma 14, this occurs with probability at most $\Pr_{\mathsf{ideal}}[\mathsf{Bad}_{4.1}]$. Therefore, except with probability at most $\Pr_{\mathsf{ideal}}[\mathsf{Bad}_{3.1}] + \Pr_{\mathsf{ideal}}[\mathsf{Bad}_{4.1}]$, the paths P and P' share either a visible node or an invisible node adjacent to a visible node.

By Corollary 6, P and P' can share at most $t-1$ consecutive nodes. Hence, if v lies in the overlap of P and P', then v must be within distance at most $t-1$ of a visible node on P, except with probability at most $\Pr_{\mathsf{ideal}}[\mathsf{Bad}_{3.1}] + \Pr_{\mathsf{ideal}}[\mathsf{Bad}_{4.1}]$. This completes the proof. □

Lemma 16. $\Pr_{\mathsf{ideal}}[\mathsf{Bad}_4] \le \Pr_{\mathsf{ideal}}[\mathsf{Bad}_{4.1}] + \Pr_{\mathsf{ideal}}[\mathsf{Bad}_{4.2}]$.

Proof. For the visibly completable path P that is being extended, consider two situations:

- Neither endpoint of P is visible.
- One endpoint of P is visible.

In the first case, by Lemma 15, the probability that there are more than $t-1$ **consecutive invisible nodes** in one direction of P is bounded by $\Pr_{\mathsf{ideal}}[\mathsf{Bad}_{4.2}]$. In addition, the probability that the adversary creates a longer visible extension is bounded by $\Pr_{\mathsf{ideal}}[\mathsf{Bad}_{4.1}]$. Since a full path contains $5t-2$ nodes, and there are at least $2t-1$ nodes outside the visibly completable path in the direction of farther endpoint, it follows that, except with probability $\Pr_{\mathsf{ideal}}[\mathsf{Bad}_{4.1}] + \Pr_{\mathsf{ideal}}[\mathsf{Bad}_{4.2}]$, there are at least t nodes that can be freshly sampled. These nodes can be taken as $\mathsf{SpecialNodes}(P)$.

In the second case, suppose one endpoint of P is visible. By Lemma 14, the adversary may have queried nodes at the other endpoint as well, but this can create at most a segment of $t-1$ visible nodes. Combined with a span of at most $t-1$ consecutive invisible nodes (except with probability $\Pr_{ideal}[\mathsf{Bad}_{4.2}]$), the total length of the affected segment in that direction is at most $2t-2$, except with probability $\Pr_{\mathsf{ideal}}[\mathsf{Bad}_{4.1}] + \Pr_{\mathsf{ideal}}[\mathsf{Bad}_{4.2}]$. Since a full path has $5t-2$ nodes and at most $4t-3$ nodes are taken(namely, t visible nodes at the closer endpoint with a span of $t-1$ invisible nodes, and $t-1$ visible nodes at the farther endpoint with a span of $t-1$ invisible nodes; see Figure 13), there remain at least $t+1$ nodes that can be freshly sampled, we can choose t of them as $\mathsf{SpecialNodes}(P)$.

Therefore, $\Pr_{\mathsf{ideal}}[\mathsf{Bad}_4] \le \Pr_{\mathsf{ideal}}[\mathsf{Bad}_{4.1}] + \Pr_{\mathsf{ideal}}[\mathsf{Bad}_{4.2}]$. □

We first bound the probability of Bad. By a union bound,

$$\begin{aligned}\Pr_{\mathsf{ideal}}[\mathsf{Bad}] &\le \Pr_{\mathsf{ideal}}[\mathsf{Bad}_1] + \Pr_{\mathsf{ideal}}[\mathsf{Bad}_2] + \Pr_{\mathsf{ideal}}[\mathsf{Bad}_3] + \Pr_{\mathsf{ideal}}[\mathsf{Bad}_4] \\ &\le O\left(\frac{q^2}{2^{n-s}}\right) + O\left(\frac{r^2 2^s q^2}{2^n}\right) + \Pr_{\mathsf{ideal}}[\mathsf{Bad}_{4.1}] + \Pr_{\mathsf{ideal}}[\mathsf{Bad}_{4.2}].\end{aligned}$$

By Lemmas 12 and 15, we have $\Pr_{\mathsf{ideal}}[\mathsf{Bad}_{4.2}] \le \Pr_{\mathsf{ideal}}[\mathsf{Bad}_{3.1}] + \Pr_{\mathsf{ideal}}[\mathsf{Bad}_{4.1}]$, and combined with

Lemma 14, we have

$$\begin{aligned}\Pr_{\mathsf{ideal}}[\mathsf{Bad}_4] &\le \Pr_{\mathsf{ideal}}[\mathsf{Bad}_{4.1}] + \Pr_{\mathsf{ideal}}[\mathsf{Bad}_{4.2}] \\ &\le \Pr_{\mathsf{ideal}}[\mathsf{Bad}_{3.1}] + 2\Pr_{\mathsf{ideal}}[\mathsf{Bad}_{4.1}] \\ &\le O\left(\frac{r^2 2^s q^2}{2^n}\right) + O\left(\frac{q}{2^{n-s}}\right).\end{aligned}$$

Therefore,

$$\begin{aligned}\Pr_{\mathsf{ideal}}[\mathsf{Bad}] &\le O\left(\frac{q^2}{2^{n-s}}\right) + O\left(\frac{r^2 2^s q^2}{2^n}\right) + O\left(\frac{q}{2^{n-s}}\right) \\ &= O\left(\frac{q^2 r^2}{2^{n-s}}\right).\end{aligned}$$

□

Theorem 7. *The construction* Cons *is indifferentiable from the ideal permutation* $\Pi^{\pm}$ *with simulator* $\mathsf{Sim}^{\pi^{\pm}}$ *with distinguishing advantage bounded by* $O(q^2r^2/2^{n-s})$.

Proof. Because $1 - \frac{P_{\mathcal{G}_{\text{ideal}}}(\mathsf{T})}{P_{\mathcal{G}_{\text{real}}}(\mathsf{T})} \le \epsilon \iff \frac{P_{\mathcal{G}_{\text{ideal}}}(\mathsf{T})}{P_{\mathcal{G}_{\text{real}}}(\mathsf{T})} \ge 1 - \epsilon$.

So we can bound:

$$\begin{aligned}\frac{P_{\mathcal{G}_{\text{ideal}}}(\mathsf{T})}{P_{\mathcal{G}_{\text{real}}}(\mathsf{T})} &= \frac{\frac{2^{NQ}}{(2^N)_Q\, 2^{nq}}}{\frac{1}{(2^n)_q}} \\ &= \frac{2^{NQ}}{(2^N)_Q} \cdot \frac{(2^n)_q}{2^{nq}} \\ &\ge \frac{(2^n)_q}{2^{nq}} \qquad (\text{since } (2^N)_Q \le 2^{NQ}) \\ &= \prod_{i=1}^{q}\left(1 - \frac{i-1}{2^n}\right) \\ &\ge \left(1 - \frac{q}{2^n}\right)^q \\ &\ge 1 - \frac{q^2}{2^n}.\end{aligned}$$

Therefore, $\frac{P_{\mathcal{G}_{\text{ideal}}}(\mathsf{T})}{P_{\mathcal{G}_{\text{real}}}(\mathsf{T})} \ge 1 - \frac{q^2}{2^n}$, which implies $\epsilon = \frac{q^2}{2^n}$.

Applying Lemma 3 with $\epsilon = \frac{q^2}{2^n}$, we obtain

$$\mathsf{Adv}_{\mathcal{A}}(\mathcal{G}_{\text{ideal}}, \mathcal{G}_{\text{real}}) \le \Pr[\mathcal{A} \circ \mathcal{G}_{\text{ideal}} \in \mathsf{Bad}] + \frac{q^2}{2^n}.$$

By Lemma 9, we have

$$\Pr[\mathcal{A} \circ \mathcal{G}_{\text{ideal}} \in \mathsf{Bad}] \le O\left(\frac{q^2 r^2}{2^{n-s}}\right).$$

Since $\frac{q^2}{2^n} \le \frac{q^2 r^2}{2^{n-s}}$, Cons is indifferentiable from an ideal N-bit permutation with distinguishing advantage $O\left(\frac{q^2 r^2}{2^{n-s}}\right)$.

□

E An Optimization to the PSI Protocol

An optimized version of our PSI protocol is given in Figure 15. Note that if we applied an ideal permutation to the C_y values (and its inverse to the $\widetilde{C}_x$ values), then the result would be identical to our original protocol (Figure 5) instantiated with our OKVS construction from Section 3.3). Thus, the optimization is equivalent to removing this instance of the ideal permutation.

Definition 15. *Let* $(\mathsf{KeyGen}, \mathit{Encaps}, \mathit{Decaps})$ *be a KEM with ciphertext space* $\mathcal{C}_\lambda$*. We say that it has* ***ciphertext density*** $\delta(\lambda)$ *if, for every possible* $(sk, pk) \leftarrow \mathsf{KeyGen}(\lambda)$*, we have*

$$\left|\{C \mid \mathit{Decaps}(sk, C) \neq \bot\}\right| \leq \delta(\lambda) \cdot |\mathcal{C}_\lambda|$$

In other words, "valid" ciphertexts comprise at most a δ*-fraction of all possible ciphertexts.*

Definition 16. *Let* $(\mathsf{KeyGen}, \mathit{Encaps}, \mathit{Decaps})$ *be a KEM with ciphertext space* $\mathcal{C}_\lambda$*. It has* ***evasive ciphertexts*** *if for every* $C \in \mathcal{C}_\lambda$*, the probability that* $\mathit{Decaps}(sk, C) \neq \bot$*, taken over the choice of* $(sk, pk) \leftarrow \mathsf{KeyGen}(\lambda)$*, is negligible in* λ*.*

Public Parameters:
cardinalities of input sets: n_A, n_B
KEM: $\mathsf{KEM} = (\mathsf{KeyGen}, \mathsf{Encaps}, \mathsf{Decaps})$ with public key space $\mathcal{P}_\lambda$ and ciphertext space $\mathcal{C}_\lambda$
OKVS: $\mathsf{OKVS}_A = (\mathsf{Encode}_A^{\mathcal{O}_A}, \mathsf{Decode}_A^{\mathcal{O}_A})$, suitable for n_A items from $\mathcal{P}_\lambda$
OKVS: $\mathsf{OKVS}_B = (\mathsf{Encode}_B, \mathsf{Decode}_B)$, suitable for n_B items from $\mathcal{C}_\lambda$
(plain OKVS without explicit ideal oracles)
H: a random oracle with range $\mathcal{C}_\lambda$

Alice (input X, $\lvert X\rvert = n_A$):		Bob (input Y, $\lvert Y\rvert = n_B$):
for $x \in X$:		
$(sk_x, pk_x) \leftarrow \mathsf{KeyGen}(\lambda)$		
$S_A = S_A \cup \{(x, pk_x)\}$		
$D_A = \mathsf{Encode}_A^{\mathcal{O}_A}(S_A)$	$\xrightarrow{D_A}$	for $y \in Y$:
		$\widetilde{pk}_y = \mathsf{Decode}_A^{\mathcal{O}_A}(D_A, y)$
		$(C_y, -) = \mathsf{Encaps}(\widetilde{pk}_y)$
		$S_B = S_B \cup \{(y, C_y + H(y))\}$
		$D_B = \mathsf{Encode}_B(S_B)$
	$\xleftarrow{D_B}$	
for $x \in X$:		
$\widetilde{C}_x = \mathsf{Decode}_B(D_B, x) - H(x)$		
if $\mathsf{Decaps}(sk_x, \widetilde{C}_x) \neq \bot$:		
$Z = Z \cup \{x\}$		
output Z		

Figure 15: Our PSI protocol, with an optimization highlighted

Lemma 17. *The optimized protocol in Figure 15 UC-securely realizes* $\mathcal{F}_{psi}$*, if* OKVS_B *is a (plain) OKVS, and all other primitives are as in Theorem 3, and furthermore the KEM has evasive ciphertexts.*

If OKVS_B *is linear with output size* $\ell(n)$*, and the KEM has ciphertext density* $\delta(\lambda)$ *and the adversary makes* q *RO queries, then the protocol achieves (upper bound on corrupt Bob's input set):*

$$\tilde{n}_B = \ell(n) \frac{\log |\mathcal{C}_\lambda| + \kappa/\ell(n)}{\log(1/\delta(\lambda)) - \log q}$$

Hybrid 2:
SimOracle plays role of $\mathcal{O}_A$

KV(x):
  if $\mathcal{V}[x]$ undefined:
    $(sk_x, pk_x) \leftarrow \mathsf{KeyGen}(\lambda)$
    $\mathcal{V}[x] = pk_x$
  return $\mathcal{V}[x]$

$D_A = \mathsf{SimOKVS}^{\mathsf{KV}}(\emptyset)$

give D_A to adversary
receive D_B from adversary

for $x \in X$:
  call KV(x) to ensure sk_x defined
  $\widetilde{C}_x = \mathsf{Decode}_B(D_B, x) - H(x)$
  if $\mathsf{Decaps}(sk_x, \widetilde{C}_x) \neq \perp$:
    $Z = Z \cup \{x\}$
output Z

Hybrid 3:
SimOracle plays role of $\mathcal{O}_A$

KV(x):
  if $\mathcal{V}[x]$ undefined:
    $(sk_x, pk_x) \leftarrow \mathsf{KeyGen}(\lambda)$
    $\mathcal{V}[x] = pk_x$
  return $\mathcal{V}[x]$

$D_A = \mathsf{SimOKVS}^{\mathsf{KV}}(\emptyset)$

give D_A to adversary
receive D_B from adversary

$$\tilde{Y} = \left\{ y \;\middle|\; \begin{array}{c} \mathcal{V}[y] \text{ defined and} \\ \mathsf{Decaps}\big(sk_y, \mathsf{Decode}_B(D_B, y) - H(y)\big) \neq \perp \end{array} \right\}$$

output $Z \cap \tilde{Y}$

Figure 16: Hybrids for the case of corrupt Bob in Lemma 17.

except with probability $1/2^{\kappa}$.

Proof. The proof of security for corrupt Alice is identical to the unoptimized protocol. It only uses the fact that OKVS_B satisfies obliviousness; adding $H(y)$ to each value in the OKVS does not undermine its obliviousness.

Thus, we focus on the case of a corrupt Bob. The first few hybrids are identical to the proof of the unmodified protocol. After a few steps we reach Hybrid 2 from this proof, which is shown in Figure 16.

Next, consider Hybrid 3 from Figure 16. The simulator extracts $\tilde{Y}$ by acting as honest Alice, under every y that was ever queried to KV. These are the values y such that the simulator knows the corresponding private key sk_y.

The hybrids have identical behavior, except in the case that (1) $\mathcal{V}[y]$ is not defined at the moment the adversary provides D_B, but (2) $\mathcal{V}[y]$ (and hence sk_y) become defined later because $y \in X$, and (3) the corresponding ciphertext $\mathsf{Decode}_B(D_B, y) - H(y)$ decapsulates without error. However, this event happens with negligible probability by a straight-forward reduction to the evasive ciphertexts property of the KEM.

It is easy to see that Hybrid 3 is equivalent to an ideal world in which the simulator sends $\tilde{Y}$ to the ideal functionality as corrupt Bob's ideal input.

Finally, we must argue that $|\tilde{Y}|$ is bounded. We do so by showing that D_B can compress the random oracle H at all points in $\tilde{Y}$.

We follow a similar outline as the proof of Lemma 2. Let $\mathcal{A}$ be an adversary that corrupts Bob in the PSI protocol and makes q queries to H. Define the following (*inefficient!*) compression algorithm:

$\underline{\mathsf{Compress}(r_1, \ldots, r_q, \mathcal{O}_A)}$:
run $\mathcal{A}$ as in Hybrid 3
use r_i to answer the adversary's ith fresh query to H
record k_i = the input to the ith query to H
answer $\mathcal{A}$'s $\mathcal{O}_A$ queries with the given oracle

when $\mathcal{A}$ outputs D_B, compute $\tilde{Y}$ as in the hybrid
abort if $|\tilde{Y}| < \beta$

output $(\mathcal{O}_A, D)$
$count := 1$
for $i = 1$ to q:
if $k_i \in \tilde{Y}$ and $count \leq \beta$:
output j such that $\mathsf{Decode}(D, k_i) - H(k_i)$ is the j'th element lexicographically of $\{C \mid \mathsf{Decaps}(sk_{k_i}, C) \neq \perp\}$
$count := count + 1$
else: output $H(k_i)$

Whenever the compression algorithm does not abort, we are able to recover all of the r_i values from its output. The approach is similar to that of Lemma 2; The decompression algorithm runs $\mathcal{A}$ and recovers the $H(k)$ values from the output stream of $\mathsf{Compress}$. When the next token of $\mathsf{Compress}$'s output is an index j, we can compute the j'th element of the set $\{C \mid \mathsf{Decaps}(sk_k, C) \neq \perp\}$ and also compute $\mathsf{Decode}(D, k)$ since D is part of $\mathsf{Compress}$'s output, then solve for $H(k)$.

Thus, we can relate the probability of $|\tilde{Y}| \geq \beta$ to the number of possible inputs/outputs of $\mathsf{Compress}$. Note that there are at most $\delta|\mathcal{V}|$ possibilities for each j, where δ is the ciphertext density of the KEM. If the OKVS is linear with output length ℓ then there are $|\mathcal{V}|^\ell$ possible values for D. Hence:

$$\begin{aligned}
\Pr[|\tilde{Y}| \geq \beta] &= \Pr_a[\mathsf{Uncompress}(\mathsf{Compress}(a)) = a] \\
&\leq \frac{|\mathcal{O}_A| \cdot |\mathcal{V}|^\ell \cdot \binom{q}{\beta} |\mathcal{V}|^{q-\beta} (\delta|\mathcal{V}|)^\beta}{|\mathcal{O}_A| \cdot |\mathcal{V}|^q} \\
&= \binom{q}{\beta} |\mathcal{V}|^\ell \delta^\beta \\
&\leq |\mathcal{V}|^\ell (q\delta)^\beta
\end{aligned}$$

To ensure $\Pr[|\tilde{Y}| \geq \beta] \leq 2^{-\kappa}$, it is enough to set

$$\begin{aligned}
& (q\delta)^\beta |\mathcal{V}|^\ell \leq 2^{-\kappa} \\
\iff\ & \beta \log(q\delta) + \ell \log |\mathcal{V}| \leq -\kappa \\
\iff\ & \beta \geq \frac{\kappa + \ell \log |\mathcal{V}|}{\log(1/\delta) - \log q}
\end{aligned}$$

Note that we require δ to be negligible, and hence $\log(q\delta)$ is negative, flipping the direction of the inequality. □

Instantiations. In Figure 17 we present concrete parameters achieved by our analysis of this protocol variant.

The Diffie-Hellman KEM variant described in Section 2.2 does not have very good density, leading to a poor bound on $\tilde{n}$. For 256-bit elliptic curves and 128-bit hash, the ciphertext density is $2^{256}/2^{384} = 1/2^{128}$.

ML-KEM-512 has much better (lower) density. Ciphertexts are strings of length 768 bytes = 6144 bits, but only 2^{256} are valid under any given key, due to the Fujisaki-Okamoto transform. Thus, the density is $1/2^{5888}$.

KEM	$\lvert\mathcal{V}\rvert$	n	ℓ	$\tilde{n}$	$\tilde{n}/n$
ECDH + hash	2^{384}	128	144	1153	9.00
		256	274	2193	8.56
		512	548	4385	8.56
		1024	1096	8769	8.56
ML-KEM-512	$2^{768\cdot 8}$	128	144	147	1.18
		256	274	281	1.29
		512	548	562	1.31
		1024	1096	1125	1.32

Figure 17: Concrete parameters for our optimized PSI protocol (Appendix E). For the given values of n, the corresponding RB-OKVS encoding has length ℓ to limit the failure probability to 2^{-40}. Then considering Lemma 17, $\tilde{n}$ is the maximum number of items extracted by the simulator (except with probability 2^{-64}) against an adversary making 2^{80} oracle queries. This is more than an honest party by a factor of $\tilde{n}/n$.

F Optimization Evaluation

This appendix evaluates the optimization described in Section 5.2 and Appendix E with non-optimized version of our protocol. The results are reported in Table 2.

The non-optimization version of our protocol employs two permutations: Alice uses party bit $b = 0$, while Bob uses $b = 1$. In our construction, this is realized by xoring $(i \parallel b)$ instead of i in each round. For HCTR2, Alice and Bob are separated using independent fixed keys (or tweaks) derived from the party bit.

The benefit of the optimization depends heavily on the permutation instantiation. For Keccak-based instantiations, the effect is significant: the speedup ranges from 1.23×–1.85× for Keccak-1600-24 and 1.13×–1.72× for Keccak-1600-12. For SNEIK, the speedup ranges from 1.01× to 1.38×, while for Feistel it ranges from 1.05× to 1.53×. For HCTR2 and the non-PQ variant, the effect is small, ranging from 0.99×–1.07× and 1.00×–1.05×, respectively. Overall, the optimization is most useful for expensive permutation instantiations such as Keccak, while its effect is much smaller for faster constructions.

n	Protocol	Total Running Time (ms)				Total Communication (KiB)
		LAN (10 Gbps)		WAN (200 Mbps)		
		Single-thread	4-thread	Single-thread	4-thread	
2^7	Mini-PSI ([39])	45.00	65.44	125.05	147.82	8.04
	VOLE-PSI[PQ OTs] [37]	213.10	217.32	1504.02	1581.52	1117.36
	LEAP[22]	474.36	-	1643.95	-	2889.05
	FHE[12]	562.15	381.66	1495.13	1347.26	733.61
	Ours [Keccak1600-24]	295.89	161.78	609.55	270.65	225.00
	Ours [Keccak1600-12]	183.02	131.85	490.88	241.92	225.00
	Ours [SNEIK]	97.99	106.03	413.42	216.86	225.00
	Ours [Feistel]	118.61	108.98	415.49	219.73	225.00
	Ours [HCTR2]	66.11	93.67	356.20	192.33	225.00
	Ours [Non-PQ]	68.35	65.55	177.72	172.77	13.50
2^8	Mini-PSI ([39])	95.23	96.68	166.50	163.43	16.04
	VOLE-PSI[PQ OTs] [37]	214.83	218.99	1589.91	1587.85	1121.14
	LEAP[22]	948.74	-	2662.90	-	5778.06
	FHE[12]	589.65	416.80	1485.46	1383.58	740.88
	Ours [Keccak1600-24]	573.76	230.79	1277.99	343.57	428.12
	Ours [Keccak1600-12]	348.77	174.03	1050.17	286.56	428.12
	Ours [SNEIK]	180.35	127.07	883.47	239.77	428.12
	Ours [Feistel]	222.46	137.23	914.09	248.03	428.12
	Ours [HCTR2]	117.46	107.06	745.77	208.85	428.12
	Ours [Non-PQ]	124.27	80.45	226.26	188.86	25.69
2^9	Mini-PSI ([39])	164.88	186.26	286.04	223.29	32.04
	VOLE-PSI[PQ OTs] [37]	225.09	229.18	1627.45	1607.12	1138.69
	LEAP[22]	1897.43	-	4452.14	-	11556.05
	FHE[12]	664.65	483.69	1619.23	1467.37	757.40
	Ours [Keccak1600-24]	1121.78	371.67	2031.64	690.07	856.25
	Ours [Keccak1600-12]	671.93	258.00	1569.57	571.44	856.25
	Ours [SNEIK]	337.03	169.19	1244.19	485.53	856.25
	Ours [Feistel]	420.82	187.44	1316.27	505.32	856.25
	Ours [HCTR2]	212.68	134.98	1108.19	437.26	856.25
	Ours [Non-PQ]	236.41	109.98	343.17	214.91	51.38
2^{10}	Mini-PSI ([39])	310.44	302.18	381.84	308.39	64.04
	VOLE-PSI[PQ OTs] [37]	232.70	238.95	1635.46	1631.83	1163.03
	LEAP[22]	3772.50	-	8039.84	-	23112.05
	FHE[12]	895.81	670.13	1911.05	1707.79	1200.77
	Ours [Keccak1600-24]	2218.26	655.77	3321.52	1367.37	1712.50
	Ours [Keccak1600-12]	1318.30	428.38	2425.05	1136.38	1712.50
	Ours [SNEIK]	645.95	254.33	1745.28	963.10	1712.50
	Ours [Feistel]	815.26	294.16	1921.29	999.87	1712.50
	Ours [HCTR2]	398.88	187.74	1476.80	878.82	1712.50
	Ours [Non-PQ]	460.01	167.66	566.28	275.59	102.75

Table 1: Performance comparison under different settings. All of our protocols use the optimization in Section 5.2.

n	Permutation / Variant	LAN (ms)				200 Mbps (ms)			
		Single-thread		4-thread		Single-thread		4-thread	
		Opt.	No opt.	Opt.	No opt.	Opt.	No opt.	Opt.	No opt.
2^7	Keccak-1600-24	295.89	528.18	161.78	237.21	609.55	821.84	270.65	332.73
	Keccak-1600-12	183.02	302.80	131.85	168.73	490.88	595.25	241.92	272.67
	SNEIK	97.99	129.56	106.03	118.11	413.42	447.01	216.86	225.92
	HCTR2	66.11	68.01	93.67	94.36	356.20	378.40	192.33	193.47
	Non-PQ	68.35	69.68	65.55	68.75	177.72	179.01	172.77	175.76
	Feistel	118.61	174.52	108.98	124.78	415.49	473.51	219.73	229.80
2^8	Keccak-1600-24	573.76	1034.72	230.79	379.77	1277.99	1712.20	343.57	467.06
	Keccak-1600-12	348.77	598.62	174.03	246.21	1050.17	1237.34	286.56	349.64
	SNEIK	180.35	242.30	127.07	147.01	883.47	957.54	239.77	261.70
	HCTR2	117.46	119.11	107.06	109.25	745.77	794.28	208.85	209.95
	Non-PQ	124.27	125.53	80.45	83.17	226.26	226.70	188.86	191.59
	Feistel	222.46	332.38	137.23	163.50	914.09	961.59	248.03	276.82
2^9	Keccak-1600-24	1121.78	2050.57	371.67	613.18	2031.64	2911.02	690.07	918.91
	Keccak-1600-12	671.93	1157.69	258.00	380.43	1569.57	2041.99	571.44	692.18
	SNEIK	337.03	460.83	169.19	204.47	1244.19	1382.37	485.53	516.11
	HCTR2	212.68	212.86	134.98	134.65	1108.19	1114.62	437.26	436.66
	Non-PQ	236.41	237.67	109.98	112.67	343.17	345.94	214.91	220.01
	Feistel	420.82	633.42	187.44	244.12	1316.27	1451.65	505.32	542.13
2^{10}	Keccak-1600-24	2218.26	4106.44	655.77	1135.46	3321.52	5119.51	1367.37	1790.83
	Keccak-1600-12	1318.30	2271.02	428.38	676.57	2425.05	3309.59	1136.38	1331.63
	SNEIK	645.95	893.13	254.33	321.13	1745.28	1932.19	963.10	975.73
	HCTR2	398.88	402.03	187.74	187.91	1476.80	1467.07	878.82	880.31
	Non-PQ	460.01	460.98	167.66	169.72	566.28	570.06	275.59	277.23
	Feistel	815.26	1249.30	294.16	398.41	1921.29	2196.24	999.87	1053.97

Table 2: Performance comparison between optimized and non-optimized versions of our protocols.