On One-Shot Signatures, Quantum vs Classical Binding, and Obfuscating Permutations

Omri Shmueli and Mark Zhandry

NTT Research

Abstract

One-shot signatures (OSS) were defined by Amos, Georgiou, Kiayias, and Zhandry (STOC'20). These allow for signing exactly one message, after which the signing key self-destructs, preventing a second message from ever being signed. While such an object is impossible classically, Amos et al observe that OSS may be possible using quantum signing keys by leveraging the no-cloning principle. OSS has since become an important conceptual tool with many applications in decentralized settings and for quantum cryptography with classical communication. OSS are also closely related to separations between classical-binding and collapse-binding for post-quantum hashing and commitments. Unfortunately, the only known OSS construction due to Amos et al. was only justified in a classical oracle model, and moreover their justification was ultimately found to contain a fatal bug. Thus, the existence of OSS, even in a classical idealized model, has remained open.

We give the first standard-model OSS, with provable security assuming (sub-exponential) indistinguishability obfuscation (iO) and LWE. This also gives the first standard-model separation between classical and collapse-binding post-quantum commitments/hashing, solving a decade-old open problem. Along the way, we also give the first construction with unconditional security relative to a classical oracle. To achieve our standard-model construction, we develop a notion of permutable pseudorandom permutations (permutable PRPs), and show how they are useful for translating oracle proofs involving random permutations into obfuscation-based proofs. In particular, obfuscating permutable PRPs gives a trapdoor one-way permutation that is *full-domain*, solving another decade-old-problem of constructing this object from (sub-exponential) iO and one-way functions.

Contents

1 Introduction

One-shot signatures (OSS) were originally proposed by Amos, Georgiou, Kiayias, and Zhandry [AGKZ20]. Here, we have a signer who generates a verification key/signing key pair, publishes the verification key, but keeps the signing key secret. Later, using the secret signing key, the signer can sign any message of their choosing. However, doing so *provably* destroys their signing key, rendering it impossible to sign even a second message relative to the same verification key. Note that the signer has full control over generating the verification key, signing key, message, and signature, and even with all this control they can never produce a second signature.

Such a protocol is clearly impossible classically. Using quantum protocols, however, it is conjectured by [AGKZ20] that an OSS is possible. Namely, the setup procedure now produces a *quantum* signing key, and the (quantum) signing algorithm is such that it requires measuring the key, which destroys it and prevents subsequent signatures. Verification keys, messages, signatures, and even the verification algorithm itself remain classical. Such an object is not trivially impossible, since the measurement principle of quantum mechanics means that computing the signature may irreversibly destroy the signing key.

OSS yields many applications that are not otherwise known: Smart contracts without a blockchain [Sat22], overcoming lower-bounds in consensus protocols, a solution to the blockchain scalability problem [CS20], quantum money with classical communication, and more. Because of these advantages, OSS has gained significant interest within the blockchain community, were a major focus of recent workshops [QSi24, NTT23], and it has been claimed that practical OSS would "completely change the endgame of blockchains" [Dra23]. See Section 1.2 for more discussion.

Unfortunately, the status of OSS has been unclear. OSS lives atop a hierarchy of notions relating to quantum money, with even milder versions being notoriously difficult to construct. [AGKZ20] observe that prior work of [ARU14, Unr16b] can be used to give OSS relative to a *quantum* oracle[1], but there was no known way to actually instantiate this oracle, even heuristically. [AGKZ20] then propose a construction relative to a *classical* oracle[2], which can then be heuristically instantiated using (post-quantum) indistinguishability obfuscation (iO). To date, this was the only plausible candidate construction.

To justify the plausibility of their construction, [AGKZ20] prove the security of their scheme in the classical oracle model. Unfortunately, their proof turned out to have a bug discovered by [Bar23], which appears fatal and the proof in [AGKZ20] has been retracted. See Section 1.3 for discussion. This bug does not indicate an actual attack, but has left the existence of OSS uncertain.

Post-quantum hashing and commitments. Even before quantum computers will enable powerful quantum protocols, they will pose a threat to current classical cryptography. Sometimes, even if the underlying building blocks are replaced with "post-quantum" equivalents, security vulnerabilities may remain. One notable example is the security of commitments: as observed by Unruh [Unr16b], the classical notion of binding for commitments is insufficient against quantum attacks. That is, even if the security game for commitments is "upgraded" to allow the adversary to run a quantum computer but is otherwise unchanged, this is not enough to guarantee meaningful security in a quantum world. Instead a stronger, inherently quantum, notion of *collapse*-binding is needed. When instantiating commitments from hashing, (post-quantum) collision-resistance is

[1] That is, an oracle that performs general unitary transformations.

[2] That is, an oracle implementing a classical function but which can be queried in superposition.

likewise not enough, and instead a stronger notion of *collapsing* is needed.

One may wonder whether classical-binding / collision-resistance actually *implies* collapse-binding / collapsing. Unruh argues that this is likely *not* the case, by noting that a prior work of [ARU14] gives counterexample relative to a *quantum* oracle. However, there is no known standard-model separation, or even a classical-oracle separation. It is entirely consistent with existing results that classical binding implies collapse-binding in a non-relativizing way. As evidence for this, known *positive* results [Unr16a, Zha22] have given collapse-binding commitments/hashes from essentially all of the same post-quantum assumptions known to imply classical binding. Nevertheless, our understanding of the relationship between classical-binding and quantum collapsing-binding is far from complete.

Interestingly, [AGKZ20, DS23] shows that a separation between classical and collapse-binding actually *implies* a one-shot signature, showing that these questions are closely linked.[3] This is in fact how the construction in [AGKZ20] works.

One-way permutations from obfuscation? The concept of cryptographically useful *program obfuscation* dates to the pioneering work of Diffie-Hellman [DH76]. Obfuscation allows for embedding cryptographic secrets into public software. The proposal of [DH76], though not phased in this language, is the following: Obfuscate a pseudorandom permutation (PRP) to get a *trapdoor* one-way permutation. More generally, obfuscation heuristically translates security using oracles into standard-model security, by actually giving out the obfuscated code of the oracles. This is exactly the idea behind the heuristic OSS construction obtained by obfuscating [AGKZ20].

It took over 30 years for cryptographically useful general-purpose obfuscation to emerge [GGH+13]. Unfortunately, by then it was shown that obfuscation cannot in general translate oracles into the standard model [BGI+01]. For example, obfuscating an arbitrary PRP provably cannot guarantee any meaningful security. Instead, the community has settled on a much weaker but precise notion of *indistinguishability obfuscation* (iO), which says that the obfuscations of programs with the same functionality are computationally indistinguishable [BGI+01].

Though much weaker than ideal, numerous techniques have been developed to use iO. Most revolve around the use of pseudorandom *functions* (PRFs). While PRFs, just like PRPs, in general cannot be obfuscated, a strengthening known as a *puncturable* PRF [KPTZ13, BW13, BGI14] actually does give provable guarantees when using iO [SW14], leading to numerous positive results.

Despite many successes, some major open questions remain. Notably, Diffie and Hellman's original proposal of obtaining a trapdoor permutation by obfuscating a PRP remains open. In fact, to the best of our knowledge, there are no known positive iO results that obfuscate PRPs to achieve *any* goal. One key challenge is that there has been no construction of a puncturable *PRP* or analogous object, and some evidence suggests that they may be *impossible* [BKW17].

Note that [BPW16, GPSZ17] construct trapdoor permutations using iO, though these permutations are not *full-domain*, i.e., where if n is the number of bits needed to represent any element in the domain (formally, the permutation domain χ is such that $\chi \subseteq \{0,1\}^n$), then we have the equality $\chi = \{0,1\}^n$. These works use a very different structure and do not simply obfuscate a PRP. Moreover, these solutions have a major drawback that the domain χ of the permutation is a sparse set in $\{0,1\}^n$ that cannot be directly sampled from nor efficiently recognized. This complicates their use in applications (see for example [CL18]) and it is unknown how to use them in

[3] The first connection between such a separation and unclonable cryptography was due to [Zha19], who shows that a separation implies the weaker object called quantum lightning. These works improved the implication to OSS.

quantum settings [MY23], where one further wants to efficiently generate quantum superpositions over subsets of the domain. An open question is if full-domain (trapdoor) OWPs are possible from iO, whether by obfuscating a PRP or by other means. There is some indication that such a OWP may in fact be impossible [AS16].

1.1 Results and Paper Organization

There are two parts to our work.

The first part proves Theorems 1 and 2. It includes an oracle construction of a non-collapsing hash function and an unconditional proof of collision resistance. A proof overview is given in Section 2 and the full proof is in Section 4.

Theorem 1. *Relative to a classical oracle, secure OSS exists.*

Similarly to the blueprint suggested in [AGKZ20], we prove Theorem 1 through separating quantum hashing notions:

Theorem 2. *Relative to a classical oracle, there exist post-quantum collision-resistant hash functions that are non-collapsing, and there exist post-quantum classically-binding commitments that are not collapse-binding.*

We do not actually know how to prove security for the oracle in [AGKZ20]. Instead, we prove Theorems 1 and 2 by a new construction. Our construction is inspired by the previous work of [AGKZ20] and a proposal made by [Bar23] at the NTT Research Quantum Money Workshop [NTT23]. We prove security through showing the random self-reducibility of our collision problem, and a sequence of reductions to simpler problems. Ultimately we show collision finding in our hash is no easier than collision finding in plain 2-to-1 random functions, which is known to be hard. This gives the first classical-oracle construction of OSS[4], and also the first such separation of classical- and collapse-binding. The construction is even the first classical-oracle construction of *quantum lightning*, a weaker notion than OSS, proposed in [Zha19].

The second part of this paper proves Theorems 3, 4, 5 and 6. It includes the development of a new cryptographic notion we call permutable pseudorandom permutations (permutable PRPs). We show how permutable PRPs can be used to prove the security of our construction in the standard model, and also for solving a number of long-standing open problems in cryptography, as elaborated below.

Theorem 3. *There exists secure OSS assuming each of the following: (1) sub-exponentially-secure indistinguishability obfuscation, (2) sub-exponentially-secure one-way functions, and (3) (polynomially-secure) LWE with a sub-exponential noise-modulus ratio.*

Theorem 4. *Under the same assumptions as Theorem 3, there exist post-quantum collision-resistant hash functions that are non-collapsing, and there exist post-quantum classically-binding commitments that are not collapse-binding.*

[4]By "classical-oracle construction," we mean the first construction *provably* and *unconditionally* secure relative to a classical oracle.

As before, Theorem 3 follows from Theorem 4, which is proved in full in Section 6. This gives the first standard-model OSS[5], and also the first standard-model separation between classical- and collapse-binding, solving this decade-old question[6]. Even for the weaker notion of quantum lightning, for which there exist a handful of candidates, this is the first construction with provable security under widely-studied assumptions.

The basic idea is to obfuscate the functions in our oracle construction rather than putting it in an oracle. The main technical gap between proving security in the oracle model and the standard model, stems from the fact that the construction uses random permutations. Our oracle construction uses a (truly) random permutation and our standard model construction uses a pseudorandom permutation (PRP). Obfuscating PRPs and getting any formal security guarantees, however, is a known challenge in cryptography, independently of quantum computation.

We develop a new notion of PRPs, called *permutable* PRPs, that can be obfuscated using iO with provable security. A permutable PRP Π very roughly allows the following: Given a key k and a (known) permutation Γ in the form of having its circuit, one can produce a "permuted" key k^{Γ}, which allows for computing $\Gamma\left(\Pi\left(k,\cdot\right)\right)$ (as well as the inverse $\Pi^{-1}\left(k,\Gamma^{-1}\left(\cdot\right)\right)$). Moreover, the key k^{Γ} hides the fact that the outputs were permuted by Γ. See Section 2 for a more detailed explanation, and Section 5 for a formal definition, as well as a formal statement and proof of the following:

Theorem 5 (Informal)**.** *There exist permutable PRPs for a large class of* Γ *assuming (1) sub-exponentially-secure iO and (2) sub-exponentially-secure one-way functions.*

Such PRPs can be seen as the PRP analogue of a puncturable pseudorandom function (PRF), which is one of the main techniques used to prove security in the iO literature. To the best of our knowledge, this is the first example which provably obfuscates a PRP. We also show that our techniques are quite general, and in Section 5 we prove the following:

Theorem 6. *There exist* full-domain *trapdoor one-way permutations (OWPs), assuming (1) sub-exponentially-secure iO and (2) sub-exponentially-secure one-way functions.*

Here, we remind that our notion of "full-domain" means that the permutation domain is just $\{0,1\}^n$. We thus solve the decade-old problem of constructing full-domain (trapdoor) one-way permutations from iO[7], and give an answer to the decades-old problem of obfuscating PRPs to obtain trapdoor permutations. In light of the impossibility in [AS16], Theorem 6 may seem surprising. However, we explain in Section 1.4 that it actually does *not* contradict their impossibility.

Our trapdoor OWP simply obfuscates a permutable PRP. The proof is straightforward given a permutable PRP; the bulk of the technical effort is then in constructing the permutable PRP in Theorem 5. This demonstrates the utility of our new PRP notion.

As a concrete application, by plugging into the elegant proof of quantumness of [MY23], we immediately obtain:

Corollary 7. *Assuming sub-exponentially (classically) secure iO and sub-exponentially (classically) secure one-way functions, there exists a proof of quantumness protocol.*

This is the first proof of quantumness using iO, as the non-full-domain trapdoor permutations of [BPW16, GPSZ17] cannot be used in this construction.

[5] By "standard-model," we mean with provable security under widely-used computational assumptions, as opposed to merely conjecturing that a construction is secure.

[6] [Unr16b] was first made public in early 2015.

[7] [BPW16] was first made public in early 2015.

Paper Organization. In the remainder of the introduction, we discuss additional motivation and related work (Section 1.2), why the bug in [AGKZ20] is unfixable (Section 1.3), and why the impossibility of [AS16] does not apply to our construction (Section 1.4). In Section 2, we provide an overview of our techniques. A reader only interested in our results on obfuscating PRPs, including our application to trapdoor permutations, can find an overview in Section 2.3. Section 2.3 is entirely classical and can be read independently of the rest of the paper without any background in quantum computing. A reader interested in our OSS construction should start with Section 2.2 which gives our oracle construction and an overview of the oracle proof. Then after developing our techniques for obfuscating PRPs, we explain how to translate our oracle construction into a standard-model construction in Section 2.4.

1.2 Motivation and Other Related Work

Quantum money and variants. Quantum money uses unclonable quantum states as currency to prevent counterfeiting. One-shot signatures lives at the top of a hierarchy of related concepts:

- **Secret key quantum money.** This was originally proposed by Wiesner [Wie83]. A major drawback, however, is that only the mint is able to verify, leading to a number of limitations.
- **Public key quantum money.** This was proposed by Aaronson [Aar09] to remedy the various issues with Wiesner's scheme. Here, anyone can verify banknotes but only the mint can create new notes. It has been a major challenge to construct public key quantum money. Several candidate constructions exist [FGH+12, KSS22, LMZ23, Zha24]. It was also shown to exist in a classical oracle model [AC12], which was later improved to a standard-model construction using iO by [Zha19].
- **Quantum lightning.** This concept allows anyone to mint banknotes together with classical serial numbers such that anyone can verify pairs of a serial number and a quantum banknote, but ensures that no user can create two valid banknotes with the same serial number. This last property of unclonability (even for the state generator) allows to use quantum lightning in decentralized settings, in ways that standard (public-key) quantum money cannot generally be used. Quantum lightning was first formalized by [Zha19], and implies in particular a public-key quantum money scheme, where a banknote is a quantum lightning state/serial number pair $|\$\rangle, \sigma$, together with a signature on σ, signed using the mint's (standard classical post-quantum) signing key. Some of the quantum money candidates are also quantum lightning [FGH+12, KSS22, LMZ23, Zha24]. But others, including the classical oracle and iO results mentioned above, are not quantum lightning. Prior to our work, all quantum lightning schemes required novel computational assumptions. Moreover, no prior scheme has provable security in a classical oracle model. In fact, our OSS gives in particular the first quantum lightning scheme with provable security under widely-studied assumptions, and an unconditional proof in a classical oracle model.
- **One-shot signatures (OSS).** Further strengthening quantum lightning, OSS treats the lightning state as a quantum signing key, which can sign a single message and then provably self-destructs. Among other things, the upgrade from quantum lightning to OSS adds the ability to use only classical communication and local quantum computation, decrease the needed coherence times for quantum lightning states, and more. No known provable classical-oracle

constructions nor standard-model instantiations (under *any* reasonable-sounding assumption) were known prior to this work.

Quantum cryptography with classical communication. An interesting application of OSS is to send quantum money using classical communication. This may sound impossible at first, but [AGKZ20] observe that it is nevertheless possible to send quantum money using a simple classical *interactive* protocol: for the mint to send a money state to Alice, Alice will create a lightning state/OSS signing key and serial number *for herself* $|\$_{\mathsf{Alice}}\rangle, \sigma_{\mathsf{Alice}}$, and send σ to the mint. The mint then signs the serial number, providing the signature $\sigma_{\mathsf{mint}\to\mathsf{Alice}}$, which the mint sends back to Alice. Now Alice's money state is $|\$_{\mathsf{Alice}}\rangle, \sigma_{\mathsf{Alice}}, \sigma_{\mathsf{mint}\to\mathsf{Alice}}$, which was obtained by just sending classical messages! Then, if Alice wants to send money to Bob, Bob simply creates a new lightning state/OSS signing key and serial number *for himself* $|\$_{\mathsf{Bob}}\rangle, \sigma_{\mathsf{Bob}}$, sends σ_{Bob} to Alice, who then signs σ_{Bob} using her quantum lightning/OSS signing key, obtaining signature $\sigma_{\mathsf{Alice}\to\mathsf{Bob}}$, which she sends to Bob. Bob now has the money state $|\$_{\mathsf{Bob}}\rangle, \sigma_{\mathsf{Bob}}, \sigma_{\mathsf{Alice}\to\mathsf{Bob}}, \sigma_{\mathsf{mint}\to\mathsf{Alice}}$. By the OSS guarantee, Alice's money state has now self-destructed, meaning she no longer has the money but Bob does. Bob can then send the money to Charlie, etc.[8]

The Mint-to-Alice step using classical communication was previously solved by [Shm22a] using iO and other tools, but the money could not be subsequently sent to Bob without quantum communication. Our work allows for Alice to send money to Bob classically, then Bob to Charlie, etc. These works on quantum money with classical communication are part of a broader class of protocols for performing quantum cryptography using classical communications, such as tests of quantumness and certified randomness [BCM+18], position verification [LLQ22], and certified deletion [BKP23, BK23, KNY23, BGK+24].

Cryptocurrencies and Blockchains. One-shot signatures have numerous applications in the cryptocurrency and blockchain settings. For example, they can be used to give decentralized currency and even smart contracts without a blockchain at all [Zha19, AGKZ20, Sat22]. OSS are known [CS20] to provide a solution to the blockchain scalability problem[9], using only classical communication. They also have various advantages for other blockchain-related tasks, such as decreasing the threshold for perfect finality, eliminating slashing and leakage risks in liquid staking, and more [Dra23].

Obfuscating pseudorandom objects. Perhaps the main technique in the literature for using iO is the punctured programming paradigm [SW14], which primarily utilizes a puncturable pseudorandom function [KPTZ13, BW13, BGI14]. These are functions where one can give out a "punctured" key, which allows for evaluating the PRF on all but a single point x. Meanwhile, even with this ability, the value on x remains pseudorandom.

Puncturable *invertible* functions were considered in [BKW17], though their construction expands the input size, so it is not a permutation but rather an (efficiently invertible) injection. They

[8]The Mint-to-Alice step actually only requires quantum lightning, but Alice-to-Bob, etc seem to require the stronger OSS.

[9]The work of [CS20] uses a strengthening of quantum lightning, where there is an additional procedure to destroy quantum lightning states and produce a classical "proof of deletion". This strengthening is known to follow from OSS, but not from standard quantum lightning.

also discuss puncturable PRPs, and explain that they are impossible in some settings. In particular, in the setting where the domain is polynomial-sized, a key that reveals the permutation on all points but x also reveals x.

Our notion of permutable PRPs avoids this issue and even is valid in the small-domain setting. We can "puncture" a permutable PRP at a point x with output y by choosing a random image y', letting Γ be the transposition which swaps y and y', and outputting the permuted key k^Γ. This allows for computing the PRP on all points except for x (and also the pre-image x' of y'), but hides the true value of y. The impossibility of punctured PRPs has come up in several settings (e.g. [SACM21, MZRA22, LMZ23, HPPY24]). It would be interesting to explore if our notion could be useful in these settings.

(Trapdoor) OWPs from iO. Trapdoor one-way permutations (OWP) were proposed by [Yao82] to abstract the ideas behind public key cryptosystems based on both RSA and Rabin. The domain and range of these objects are sets with algebraic structure. However, it became much simpler to describe applications in terms of a full-domain OWP where the domain and range are simply $\{0,1\}^n$. This simplification lead to several conceptual errors (see [GR13] for explanation). To account for these errors, when using sparse-domain OWPs, one must often stipulate additional conditions (called "enhanced" or "doubly enhanced") that are trivially satisfied by full-domain OWPS.

Once iO emerged as a powerful cryptographic tool [GGH+13], a natural question was whether iO could give a third way of building a trapdoor permutation. A particular motivation is to achieve post-quantum security, since RSA and Rabin are both quantumly insecure due to Shor's algorithm [Sho94]. Constructions were given [BPW16, GPSZ17], but these constructions were "messy," with sparse domains that could not be efficiently recognized and required cryptographic procedures to sample. An interesting question was whether a clean, full-domain trapdoor permutation was possible from iO. Our work mostly resolves this question, though we still have the limitation that the space of keys is sparse.

As a concrete application, our trapdoor OWPs can be plugged into the elegant proofs of quantumness of [MY23], to obtain proofs of quantumness from (classically-hard sub-exponential) iO and one-way functions[10]. This is the first proof of quantumness based on iO. The prior OWPs of [BPW16, GPSZ17], despite being doubly enhanced, did not suffice for this application. More generally, permutations on simple domains like $\{0,1\}^n$ appear much more useful in the context of quantum cryptosystems, whereas permutations on highly structured domains break the delicate structure of quantum states.

1.3 The Bug in [AGKZ20]

The bug in [AGKZ20] as found by [Bar23] is rather technical, and we do not discuss it here. However, we argue that the bug is likely unfixable using the techniques employed by [AGKZ20]. The issue is that [AGKZ20] use what is known as the inner-product adversary method – first developed in [AC12] – to prove the collision resistance of a certain hash function. Unfortunately, the inner-product adversary method is unlikely to be able to prove the basic collision problem is hard, even before adding all the structure that [AGKZ20] need to obtain their OSS scheme. The reason is that the collision problem has a small *certificate complexity*, and it is known that the

[10]The OWP part of our work does not need LWE.

adversary method cannot prove good lower-bounds for problems with small certificate complexity, as shown by [Aar03]. This certificate complexity barrier extends to all "positive weight" adversary methods, including the inner-product method. Thus, it seems any fixed proof for [AGKZ20] must use additional techniques.[11]

Note that most of our oracle result is proved using standard adversary method techniques. However, our result assumes the standard collision lower bound as previously proved in [AS04, Zha15]. These results were proved using the polynomial method that is not subject to the certificate complexity barrier.

1.4 On the Lower Bound for OWPs from iO

Asharov and Segev [AS16] show a barrier for constructing OWPs from iO and one-way functions. Specifically, they show that any "black-box" construction of one-way permutations from iO and one-way functions, cannot be "domain-invariant". Here, black-box in the context of iO is a bit subtle, but their notion captures most known iO techniques, including ours. Recall that in a (keyed) OWP, we sample one permutation out of exponentially many permutations in an efficiently samplable family. As for domain-invariance, the abstract of [AS16] defines it as "each permutation [in the family] may have its own domain, but these domains are independent of the underlying building blocks". At first glance, this would seem to contradict our full-domain trapdoor permutation, since the domain of all permutations in the family of our construction is just $\{0,1\}^n$, clearly independent of any building block.

However, there are really two types of domain invariance: One for the key, and one for the actual input. The formal specification of the impossibility in [AS16] reveals that their notion of domain-invariance refers to schemes that satisfy *both* notions. In fact, in the model they propose, obfuscating a PRP actually does yield an *input*-domain-invariant (trapdoor) OWP; also, since the key is an obfuscated program and most strings do not correspond to valid programs / keys, it is not key-domain invariant.

Our (trapdoor) OWPs are similarly input-domain invariant but not key-domain invariant, since our key is likewise an obfuscated program, with the added benefit of provable security under iO. Thus, our result does not contradict the impossibility of [AS16].

1.5 Directions for Future Work

Here are some natural follow-up directions from our work:

- Can OSS be achieved without using iO? While there are several approaches (using new hardness assumptions) that seem to give weaker objects like quantum lightning [FGH+12, KSS22, LMZ23, Zha24], they do not seem amenable to producing classical signatures, even heuristically.

- Is it possible to remove the need for sub-exponential hardness in our constructions? Sub-exponential hardness arises in two key places. The first is in our PRP: we start with a "base"

[11] One caveat is that the barrier only applies to lower bounds for the number of queries to distinguish a function with collisions from a function that is injective. Such a lower-bound immediately implies a lower-bound for actually finding collisions. But the converse is not true, and the certificate complexity barrier does *not* seem to rule out directly proving the hardness of finding collisions. Nevertheless, it does not seem like the techniques of [AGKZ20] circumvent this barrier.

PRP (based just on one-way functions) which is permutable only for the class of "neighbor swaps" that exchange a given z with $z+1$, and upgrade it to more general permutations Γ by decomposing such permutations into a sequence of neighbor swaps. The proof incurring a security loss for each step in the decomposition. General permutations, even just transpositions (which are enough to get trapdoor OWPs) require an exponential-sized decomposition. In order to get around this limitation, it seems that the "base" PRP needs to support a richer class than just neighbor swaps. Is there a base PRP that support, say, general transpositions, from just one-way functions? The second place sub-exponential hardness comes in is when we are switching from the construction to a simulated distribution; this utilizes a hybrid over all possible outputs. Perhaps there is a clever way to change all possible outputs in one go.

- Can we achieve a "clean iO" approach to OSS that uses just iO and generic primitives? Note that it is unlikely that OSS can be obtained from iO and one-way functions: OSS requires collision resistance, which is believed to not be possible from iO and one-way functions alone [AS15]. But perhaps iO and collision-resistance is enough. Or at a minimum, maybe our LWE assumption can be replaced with other algebraic techniques to give a diversity of assumptions.

- We introduced a new technique for obfuscating PRPs, and it would be interesting to see if this enables any new results, or at least streamlines old results.

Finally, we conclude with a fascinating complexity-theoretic question inspired by our techniques. Any permutation Γ on $\{0,1\}^n$ can be decomposed into an exponentially long product of transpositions $\Gamma = (a_1 \;\; a_2) \circ (a_3 \;\; a_4) \circ \cdots$. While such a decomposition is clearly inefficient, the following question asks if the partial products can be made small:

Question 8 (Efficient Permutation Decomposition Problem)**.** *For any permutation Γ on $\{0,1\}^n$ such that Γ, Γ^{-1} have circuits of size s, is it possible to decompose Γ into a product of transpositions $\Gamma = \tau_1 \circ \tau_2 \circ \cdots \circ \tau_T$ such that each partial product $\Gamma_t := \tau_1 \circ \tau_2 \circ \cdots \circ \tau_t$ for $t \in [T]$ and its inverse Γ_t^{-1} have circuits of size $\mathsf{poly}(s,n)$?*

As part of constructing our permutable PRPs and applying them to construct OSS, we show that very general families of permutations can be efficiently decomposed in this way (see Figure 5 for a non-exhaustive list of such families). On the other hand, we do not know how to handle all efficient permutations, and even for very simple permutations like multiplying by a scalar mod N, we only know a general solution assuming the Extended Riemann Hypothesis.

2 Technical Overview

In this section we explain the main techniques shown in this work.

2.1 Definitions

We first give the formal definition of one-shot signatures.

Definition 9 (One-Shot Signature Scheme)**.** *A one-shot signature (OSS) scheme is a tuple of algorithms $(\mathsf{Setup}, \mathsf{Gen}, \mathsf{Sign}, \mathsf{Ver})$ together with message-space $(\mathcal{M}_\lambda)_\lambda$ satisfying the following:*

- $\mathsf{CRS} \leftarrow \mathsf{Setup}\left(1^\lambda\right)$: *A classical probabilistic polynomial-time algorithm that given the security parameter, samples the classical common reference string (CRS).*
- $(\mathsf{pk}, |\mathsf{sk}\rangle) \leftarrow \mathsf{Gen}(\mathsf{CRS})$: *A quantum polynomial-time algorithm that takes the classical* CRS *and samples a classical public key* pk *and quantum secret key* $|\mathsf{sk}\rangle$.
- $\sigma \leftarrow \mathsf{Sign}\left(\mathsf{CRS}, |\mathsf{sk}\rangle, m\right)$: *A quantum polynomial-time algorithm that given* CRS *and the quantum key* $|\mathsf{sk}\rangle$*, given any message* $m \in \mathcal{M}_\lambda$ *produces a classical signature* σ.
- $\mathsf{Ver}\left(\mathsf{CRS}, \mathsf{pk}, m, \sigma\right) \in \{0,1\}$: *A classical deterministic polynomial-time algorithm which verifies a message* m *and signature* σ *relative to a public key* pk.
- **Correctness:** *There exists a negligible function* negl *such that, for any* λ *and any* $m \in \mathcal{M}_\lambda$ *we have*
$$\Pr_{\substack{\mathsf{CRS}\leftarrow\mathsf{Setup}\left(1^\lambda\right),\\ (\mathsf{pk},|\mathsf{sk}\rangle)\leftarrow\mathsf{Gen}(\mathsf{CRS}),\\ \sigma_m\leftarrow\mathsf{Sign}(\mathsf{CRS},|\mathsf{sk}\rangle,m)}}\left[\mathsf{Ver}\left(\mathsf{CRS}, \mathsf{pk}, m, \sigma_m\right) = 1\right] \geq 1 - \mathsf{negl}\left(\lambda\right) \quad .$$
- **Security:** *For any QPT algorithm* $\mathcal{A}$*, there exists a negligible function* negl *such that for all* λ*,*
$$\Pr_{\substack{\mathsf{CRS}\leftarrow\mathsf{Setup}(1^\lambda),\\ (\mathsf{pk},m_0,m_1,\sigma_0,\sigma_1)\leftarrow\mathcal{A}(\mathsf{CRS})}}\left[\mathsf{Ver}\left(\mathsf{CRS}, \mathsf{pk}, m_0, \sigma_0\right) = 1 \ \wedge \ \mathsf{Ver}\left(\mathsf{CRS}, \mathsf{pk}, m_1, \sigma_1\right) = 1\right] \leq \mathsf{negl}\left(\lambda\right) \quad .$$

It is straightforward to adapt the above definition to utilize an oracle. In this case, the oracle will play the role of CRS, and we will omit the algorithm Setup.

OSS from Non-collapsing Hashing. As observed informally by [AGKZ20], a collision-resistant but non-collapsing hash function gives an OSS, and this was made formal and general by [DS23]. Our OSS therefore will be built from such a hash function. Here, we give the notion of collision resistant and non-collapsing hash functions.

Definition 10 (Collision-Resistant Always-Non-Collapsing Hash)**.** *A collision-resistant always-non-collapsing hash function is a pair of PPT algorithms* (Setup, H) *such that*

- $\mathsf{CRS} \leftarrow \mathsf{Setup}\left(1^\lambda\right)$: *A classical probabilistic polynomial-time algorithm that given the security parameter, samples the classical common reference string (CRS).*
- $y \leftarrow H\left(\mathsf{CRS}, x\right)$: *A classical deterministic polynomial-time algorithm that given the CRS and input* x*, outputs* y.
- **Collision-resistance:** *For any QPT algorithm* $\mathcal{A}$*, there exists a negligible function* negl *such that for all* λ*,*
$$\Pr_{\substack{\mathsf{CRS}\leftarrow\mathsf{Setup}\left(1^\lambda\right),\\ (x_0,x_1)\leftarrow\mathcal{A}(\mathsf{CRS})}}\left[H\left(\mathsf{CRS}, x_0\right) = H\left(\mathsf{CRS}, x_1\right)\right] \leq \mathsf{negl}\left(\lambda\right) \quad .$$

- **Always non-collapsing:** *There exists a pair of QPT algorithms* $(\mathcal{S}, \mathcal{D})$ *and a negligible function* $\mathsf{negl}(\lambda)$ *such that for all* λ*,*

$$\left|\Pr\left[\mathcal{D}(x, \mathsf{aux}) = 1 \ : \ \begin{smallmatrix}\mathsf{CRS}\leftarrow\mathsf{Setup}(1^\lambda),\\ (|\psi\rangle,\mathsf{aux})\leftarrow\mathcal{S}(\mathsf{CRS}),\\ x\leftarrow\mathsf{Measure}(|\psi\rangle)\end{smallmatrix}\right] - \Pr\left[\mathcal{D}(|\psi_y\rangle, \mathsf{aux}) = 1 \ : \ \begin{smallmatrix}\mathsf{CRS}\leftarrow\mathsf{Setup}(1^\lambda),\\ (|\psi\rangle,\mathsf{aux})\leftarrow\mathcal{S}(\mathsf{CRS}),\\ |\psi_y\rangle\leftarrow\mathsf{PartialMeasure}_{H(\mathsf{CRS},\cdot)}(|\psi\rangle)\end{smallmatrix}\right]\right| \geq 1 - \mathsf{negl}(\lambda) \ .$$

Here, $x \leftarrow \mathsf{Measure}(|\psi\rangle)$ *means to measure* $|\psi\rangle$ *in the computational basis arriving at measurement* x*.* $|\psi_y\rangle \leftarrow \mathsf{PartialMeasure}_{H(\mathsf{CRS},\cdot)}$ *means to compute* $H(\mathsf{CRS}, \cdot)$ *(in superposition, with output register on the side) with input register* $|\psi\rangle$*, and measuring the result, resulting in outcome* y*. Then the state* $|\psi\rangle$ *collapses to* $|\psi_y\rangle$*, which contains some superposition of preimages of* y*.*

We will not formally give the proof that such a hash function implies OSS, but give the sketch for the interested reader. The idea is that Gen runs $\mathcal{S}$ to get $|\psi\rangle, \mathsf{aux}$, and then applies $H(\mathsf{CRS}, \cdot)$ to $|\psi\rangle$ and measures, obtaining y. We set $\mathsf{pk} = y$, and $|\mathsf{sk}\rangle = |\psi_y\rangle, \mathsf{aux}$ where $|\psi_y\rangle$ is the post-measurement state. Observe that the support of $|\psi_y\rangle$ are strings x that hash to y.

For a bit b, let $|\psi_{y,b}\rangle$ be the state post-selecting on x whose first bit is b. One can create $|\psi_{y,b}\rangle$ for a random choice of b my simply measuring the first qubit. Then very roughly it is shown in [AGKZ20, DS23] how to use the distinguisher $\mathcal{D}$ to move back and forth between $|\psi_{y,0}\rangle$ and $|\psi_{y,1}\rangle$.

To sign a bit b, one obtains $|\psi_{y,b}\rangle$ and measures, obtaining a string x beginning with the bit b that hashes to y. This is a signature on b, which is easy to verify. Moreover, the collision-resistance of H implies that it is computationally infeasible to find two signatures. Thus, we have an OSS for a single bit. Then by parallel repetition, we obtain an OSS for arbitrary messages.

2.2 OSS Relative to a Classical Oracle (Section 4)

Since OSS follows from non-collapsing collision-resistant hash functions, we will focus on constructing the latter.

The Construction From [AGKZ20]. The OSS of [AGKZ20] utilizes what we will call a *coset partition function* in this work. This is a function Q that is many-to-1, and where the pre-image set of any image is a coset of a linear subspace: that is, the pre-image sets have the form $Q^{-1}(y) := \{\mathbf{A}_y \cdot \mathbf{r} + \mathbf{b}_y \ : \ \mathbf{r} \in \mathbb{Z}_2^k\}$, where $\mathbf{A}_y, \mathbf{r}_y$ are a matrix/vector pair that depends on the image y. The function Q will be the hash function, provided as an oracle.

In order to be non-collapsing, [AGKZ20] employ the hidden subspaces approach of [AC12], and additionally provide a separate oracle D, which provides membership testing for the linear space $\mathbf{A}^\perp := \{\mathbf{z} \ : \ \mathbf{A}_y \cdot \mathbf{z} = 0\}$. This allows for testing if a state is in the uniform superposition $|Q^{-1}(y)\rangle := \sum_{x \in Q^{-1}(y)} |x\rangle$: first use Q to test that the state is in the support of $Q^{-1}(y)$, then apply the quantum Fourier transform (QFT), and use D to test that the resulting state has support on $\mathbf{A}^\perp$. The only state that passes both verifications is the state $|Q^{-1}(y)\rangle$.

In order to obtain a non-collapsing hash/OSS scheme, then one needs to prove that Q is collision-resistant, given the oracles for Q and D. As mentioned, the proof provided in [AGKZ20] contained a fatal bug, and while the scheme is plausibly collision-resistant, it remains unclear how to prove this.

Relaxing the structure. A key challenge with the approach in [AGKZ20] is that it is non-trivial come up with a coset partition function that is not trivially insecure. The cosets need to have very different orientations ($\mathbf{A}_y$ values), lest the function Q be periodic and therefore subject to quantum period-finding algorithms [Sho94]. But how do we perfectly partition the domain into cosets that are all of different orientations, but no overlaps? The coset partition function used in [AGKZ20] is derived in a very specific way, and the various pre-image sets are highly correlated, making reasoning about them quite challenging. This is further complicated by the D oracle, which contains even more information about the cosets. An interesting proposal was made by [Bar23]: Simply have the hash function be a random function H, but provide an additional oracle which maps each pre-image set $H^{-1}(y)$ to a random coset S_y embedded in a much larger space. The point is that, the cosets S_y for different y's no longer need to partition the space they live in, as they are independent of each other.

Our following construction is inspired by the above general principles. We sample a random secret permutation Π on $\{0,1\}^n$, and let $H(x), J(x)$ denote the first r bits and last $n-r$ bits of $\Pi(x)$, respectively. $H : \{0,1\}^n \to \{0,1\}^r$ will be our hash function. For each $y \in \{0,1\}^r$, we also choose a random coset $S_y \subseteq \{0,1\}^k$ of dimension $n-r$, described by a matrix/vector pair $\mathbf{A}_y \in \mathbb{Z}_2^{k\times(n-r)}, \mathbf{b}_y \in \mathbb{Z}_2^k$ as $S_y = \{\mathbf{A}_y \cdot \mathbf{r} + \mathbf{b}_y : \mathbf{r} \in \{0,1\}^{n-r}\}$. We then provide an oracle $\mathcal{P}(x)$ which outputs $(H(x), \mathbf{u} = \mathbf{A} \cdot J(x) + \mathbf{b})$; in other words, it computes $H(x)$ and also the point in that coset that x maps to. We likewise provide the oracle $\mathcal{D}(y, \mathbf{v})$ which checks if $\mathbf{v} \in S_y^\perp$, where $S_y^\perp$ is the kernel of $\mathbf{A}_y$.

We have to provide one more oracle, which is denoted $\mathcal{P}^{-1}(y, \mathbf{u})$, which inverts the oracle $\mathcal{P}$: it outputs x if $\mathcal{P}(x) = (y, \mathbf{u})$ and otherwise outputs a special symbol $\perp$ indicating that $(y, \mathbf{u})$ is not in the range of $\mathcal{P}$. This oracle is necessary in order to actually preserve the non-collapsing property: given the uniform superposition of pre-images $|H^{-1}(y)\rangle$, applying the oracle $\mathcal{P}$ gives $\sum_{x\in H^{-1}(y)} |x, \mathcal{P}(x)\rangle = \sum_{x\in H^{-1}(y)} |x, y, \mathbf{A}_y \cdot J(x) + \mathbf{b}_y\rangle$. We would like to apply the QFT to the register containing $\mathbf{A}_y \cdot J(x) + \mathbf{b}_y$, but this will not work: since the register x remains around and is entangled with the register containing $\mathbf{A}_y \cdot J(x) + \mathbf{b}_y$, performing the QFT will actually yield random junk. The oracle $\mathcal{P}^{-1}$ allows us to un-compute x, thereby allowing verification to work as desired.

We now need to prove the collision-resistance of H. Unfortunately, quantum query lower-bounds for oracles with inverses is a notorious challenging problem. This problem together with the presence of the $\mathcal{D}$ oracle were the primary barriers to proving the security of this construction.

In what follows, we describe our proof in the oracle setting. At a very high-level, our proof will gradually eliminate parts of the oracles $\mathcal{P}, \mathcal{P}^{-1}, \mathcal{D}$ until all that remains is an a plain hash function oracle, and we can then invoke the known query lower-bounds for collision-finding to conclude security.

Warm-up: A Random Self-Reduction. As a warm-up that will lead to our proof, we introduce a random self-reduction for the oracles $\mathcal{P}, \mathcal{P}^{-1}, \mathcal{D}$. Given an instance of the oracles $\overline{\mathcal{P}}, \overline{\mathcal{P}^{-1}}, \overline{\mathcal{D}}$ (with underlying permutation $\overline{\Pi}$ on $\{0,1\}^n$ and matrix/vector pairs $\overline{\mathbf{A}}_y \in \mathbb{Z}_2^{k\times(n-r)}, \overline{\mathbf{b}}_y \in \mathbb{Z}_2^k$ for all $y \in \{0,1\}^r$), we can construct another instance as follows. Choose a random permutation Γ on $\{0,1\}^n$ and for every $y \in \{0,1\}^r$, choose a random full-rank $\mathbf{C}_y \in \{0,1\}^{k\times k}$ and random $\mathbf{d}_y \in \{0,1\}^k$. Note that $\mathbf{C}_y, \mathbf{d}_y$ define a random affine permutation $L_y(\mathbf{u}) = \mathbf{C}_y \cdot \mathbf{u} + \mathbf{d}_y$ on $\{0,1\}^k$. Then define:

- $\mathcal{P}(x)$: Compute $\overline{x} \leftarrow \Gamma(x)$, $(y, \overline{\mathbf{u}}) \leftarrow \overline{\mathcal{P}}(\overline{x})$, $\mathbf{u} \leftarrow L_y(\overline{\mathbf{u}})$ and output $(y, \mathbf{u})$.

- $\mathcal{P}^{-1}(y, \mathbf{u})$: Compute $\overline{\mathbf{u}} \leftarrow L_y^{-1}(\mathbf{u})$, $\overline{x} \leftarrow \overline{\mathcal{P}}^{-1}(y, \overline{\mathbf{u}})$, $x \leftarrow \Gamma^{-1}(\overline{x})$ and output x.
- $\mathcal{D}(y, \mathbf{v})$: Output $\overline{\mathcal{D}}\left(y, \mathbf{C}^T \cdot \mathbf{v}\right)$.

The resulting oracle implicitly sets $\Pi := \overline{\Pi} \circ \Gamma$, and for every $y \in \{0,1\}^r$ sets $\mathbf{A}_y := \mathbf{C}_y \cdot \overline{\mathbf{A}}_y$, $\mathbf{b}_y := \mathbf{C}_y \cdot \overline{\mathbf{b}}_y + \mathbf{d}_y$, which all distribute independently of the underlying $\overline{\Pi}$, $\overline{\mathbf{A}}_y$, $\overline{\mathbf{b}}_y$. Thus this random self-reduction turns any instance $\overline{\mathcal{P}}, \overline{\mathcal{P}^{-1}}, \overline{\mathcal{D}}$ into a fresh independent instance $\mathcal{P}, \mathcal{P}^{-1}, \mathcal{D}$. Moreover, if we let H and $\overline{H}$ be the functions outputting the first r bits of $\mathcal{P}$ and $\overline{\mathcal{P}}^{-1}$ respectively, we can turn a collision for H into a collision for $\overline{H}$ by applying Γ. Thus, we can derive the collision-resistance for random instances based on the collision resistance of any fixed instance. We note that there does not appear to be an analogous random self-reduction for the oracles of [AGKZ20]. We will use variants of this self-reduction to gradually remove components from our oracle.

Step 1: Bloating the Dual. The first step of our actual proof, inspired by techniques of [Zha19], is to "bloat the dual." This means, for each y, we choose a random super-space $T_y^\perp$ of $S_y^\perp := \mathsf{ColSpan}(\mathbf{A}_y)^\perp$, and replace $\mathcal{D}$ with the oracle $\mathcal{D}'$ which checks for membership in $T_y^\perp$. $T_y^\perp$ is chosen at random such that (1) it is a sparse subset of $\{0,1\}^k$ and (2) $S_y^\perp$ is a sparse subset of $T_y^\perp$. We can prove that $\mathcal{D}'$ is indistinguishable form $\mathcal{D}$, since the points in $T_y^\perp$ but not in $S_y^\perp$ are random sparse points hidden from the adversary. Thus an adversary that finds a collision given the more informative $\mathcal{D}$, will also find a collision given $\mathcal{D}'$.

In [Zha19], this technique was employed to reduce the security of the quantum money scheme of [AC12] to an information-theoretic statement. Crucially in that proof, *both* the primal S and dual $S^\perp$ were bloated, which then implies that the original space S is information-theoretically hidden. In our case, however, the original subspaces S_y are still part of the oracles $\mathcal{P}, \mathcal{P}^{-1}$, and it is unclear how to bloat them. This key difference is that in [Zha19], there was only a single sparse space S, and it could be bloated by consuming some of the abundant "free" ambient space. However, in our case, the entire domain is partitioned into sets $H^{-1}(y)$, and it seems that we would need to bloat each of them. But this is impossible, since there is no "free" space left as each point is already part of some subspace. Nevertheless, in the following, we will see that the dual-bloating is still useful.

Step 2: Simulating the Dual. We will now completely eliminate the dual oracle $\mathcal{D}'$. To do so, we use a version of the random self-reduction described above, but starting from a smaller instance $(\overline{\mathcal{P}}, \overline{\mathcal{P}}^{-1}, \overline{\mathcal{D}})$. We will show that embedding the smaller instance lets us (without querying the smaller instance) know some information about the subspaces $S_y^\perp$, namely a random super-space $T_y^\perp$ of $S_y^\perp$. This allows us to simulate $\mathcal{D}'$ *without querying* $\overline{\mathcal{D}}$ *at all*, and in turn to eliminate the dual oracle $\overline{\mathcal{D}}$ entirely.

In more detail, let $s = \log_2(|T_y^\perp|/|S_y^\perp|)$, which is how much larger (in terms of dimension) $T_y^\perp$ is relative to $S_y^\perp$. Let $\overline{\mathcal{P}}, \overline{\mathcal{P}}^{-1}$ be an instance of our oracle, but with input space $\{0,1\}^{n-(n-r-s)} = \{0,1\}^{r+s}$ and output space $\{0,1\}^r \otimes \{0,1\}^{k-(n-r-s)}$. In other words, we shrink the input x and the vector part of the output $\mathbf{u}$ each by $n-r-s$ bits, but we keep the y part of the output (the actual hash function output) the same length. We will not have access to the oracle $\overline{\mathcal{D}}$.

We simulate $\mathcal{P}, \mathcal{P}^{-1}, \mathcal{D}'$ using $\overline{\mathcal{P}}, \overline{\mathcal{P}}^{-1}$. However, since $\overline{\mathcal{P}}, \overline{\mathcal{P}}^{-1}$ is a smaller instance, we first expand it into a full-sized instance (i.e., without considering how our mapping distributes). Essen-

tially, we just pass the first $r+s$ bits of x into $\overline{\mathcal{P}}$, and the last $n-r-s$ bits we output in the clear as part of $\mathbf{u}$. In more detail:

- $\mathcal{P}(x)$: Break the input $x \in \{0,1\}^n$ into two parts: $x := \left(\overline{x} \in \mathbb{Z}_2^{r+s}, \widetilde{x} \in \mathbb{Z}_2^{n-r-s}\right)$ and we moreover interpret $\widetilde{x}$ as a vector of dimension $n-r-s$. Now query $(y, \overline{\mathbf{u}}) \leftarrow \overline{\mathcal{P}}(\overline{x})$, and set $\mathbf{u} = (\overline{\mathbf{u}}, \widetilde{x})$. Output $(y, \mathbf{u})$.
- $\mathcal{P}^{-1}(y, \mathbf{u})$: Write $\mathbf{u} = \left(\overline{\mathbf{u}} \in \mathbb{Z}_2^{k-(n-r-s)}, \widetilde{x} \in \mathbb{Z}_2^{n-r-s}\right)$, compute $\overline{x} \leftarrow \overline{\mathcal{P}}^{-1}(y, \overline{\mathbf{u}})$ and output $x = (\overline{x}, \widetilde{x})$.
- $\mathcal{D}'(y, \mathbf{v})$: Write $\mathbf{v} = \left(\overline{\mathbf{v}} \in \mathbb{Z}_2^{k-(n-r-s)}, \widetilde{x} \in \mathbb{Z}_2^{n-r-s}\right)$. Output 1 if and only if $\widetilde{x} = 0^{n-r-s}$.

Now this clearly does not simulate the correct distribution of oracles $\mathcal{P}, \mathcal{P}^{-1}, \mathcal{D}'$ since for example the last $n-r-s$ bits of input are clearly visible in the output, and $\mathcal{D}'$ checks a fixed subspace. However, it is a valid instance of the oracles $\mathcal{P}, \mathcal{P}^{-1}, \mathcal{D}'$, in the sense that there is *some* choice permutation Π, cosets S_y and spaces $T_y^{\perp}$ that gives these oracles, or in other words, we output an oracle that's inside the support of $\mathcal{P}, \mathcal{P}^{-1}, \mathcal{D}'$. Note that our output oracle satisfies that for every $y \in \{0,1\}^r$ we have that $T_y^{\perp}$ is just the space of vectors whose last $n-r-s$ entries are 0 – we'll use this fact later. We can then apply our random self-reduction to generate a correctly distributed instance.[12]

Let $\overline{H}$ the hash function in the oracles $\overline{\mathcal{P}}, \overline{\mathcal{P}}^{-1}$, let H the hash function in the oracles $\left(\mathcal{P}, \mathcal{P}^{-1}, \mathcal{D}'\right)$ and let $\widetilde{H}$ the hash function in the oracles $\left(\widetilde{\mathcal{P}}, \widetilde{\mathcal{P}^{-1}}, \widetilde{\mathcal{D}}\right)$, which are generated by applying the random self reducibility on $\left(\mathcal{P}, \mathcal{P}^{-1}, \mathcal{D}'\right)$. By the random self-reducibility, we know that any algorithm which finds collisions for $\widetilde{H}$ given access to $\left(\widetilde{\mathcal{P}}, \widetilde{\mathcal{P}^{-1}}, \widetilde{\mathcal{D}}\right)$ will find collisions for the simulated H relative to $\left(\mathcal{P}, \mathcal{P}^{-1}, \mathcal{D}'\right)$. To complete our elimination of the dual oracle, we need to show that such collisions in H actually yield collisions in $\overline{H}$. For simplicity, we will work with the non-random-self reduced oracles described above $\left(\mathcal{P}, \mathcal{P}^{-1}, \mathcal{D}'\right)$. First, observe that if we write $x = \left(\overline{x} \in \mathbb{Z}_2^{r+s}, \widetilde{x} \in \mathbb{Z}_2^{n-r-s}\right)$, then $H(x) = \overline{H}(\overline{x})$. There are two ways a collision x, x' in H can occur:

- $\overline{x} \neq \overline{x}'$. In this case $\overline{x}$ and $\overline{x}'$ form a collision in $\overline{H}$, as desired.
- $\overline{x} = \overline{x}'$. We will call these "bad" collisions, which we will now handle. Observe that in these cases, since $x \neq x'$, we must have $\widetilde{x} \neq \widetilde{x}'$. Also, since $\overline{x} = \overline{x}'$, we have that $\overline{\mathbf{u}} = \overline{\mathbf{u}}'$. But letting $\mathbf{u} = \left(\overline{\mathbf{u}} \in \mathbb{Z}_2^{k-(n-r-s)}, \widetilde{x} \in \mathbb{Z}_2^{n-r-s}\right)$ and $\mathbf{u}' = \left(\overline{\mathbf{u}} \in \mathbb{Z}_2^{k-(n-r-s)}, \widetilde{x}' \in \mathbb{Z}_2^{n-r-s}\right)$, this means that $\mathbf{u} - \mathbf{u}'$ is a non-zero vector whose first $k-(n-r-s)$ entries are 0.

  In this case, let T_y be the linear space that is dual to $T_y^{\perp}$. Recall that in the non-re-randomized case, $T_y^{\perp}$ is the space of vectors whose last $n-r-s$ entries are 0; this means that T_y is the space of vectors whose *first* $k-(n-r-s)$ entries are 0. Thus, "bad" collisions give $(\mathbf{u} - \mathbf{u}') \in T_y$. This property is moreover preserved by applying the random self-reduction. In contrast, a general collision will have $\mathbf{u} - \mathbf{u}'$ in a much larger space, namely S_y, the analogous linear space dual to $S_y^{\perp}$.[13]

[12] Note that in our warm-up, the random self-reduction re-randomized an instance of the actual construction $(\mathcal{P}, \mathcal{P}^{-1}, \mathcal{D})$, but we are now re-randomizing an instance $(\mathcal{P}, \mathcal{P}^{-1}, \mathcal{D}')$, when the dual is bloated. The same random self-reduction works just as well in this setting.

[13] S_y contains, and is s dimensions larger than T_y, since $S_y^{\perp}$ is contained, and is s dimensions smaller than $T_y^{\perp}$.

Thus, we see that bad collisions cause $\mathbf{u} - \mathbf{u}'$ to concentrate in a much smaller space than general collisions. Following a technique introduced in [Shm22b] (which we improve in Lemmas 22 and 26), we will use this fact to show that an algorithm which produces bad collisions actually distinguishes the bloated from un-bloated cases, contradicting our earlier proof of the indistinguishability of these two cases.

In slightly more detail, this is because we can run an adversary several times (using a separate random self-reduction each time) to collect several independent vectors $\mathbf{u} - \mathbf{u}'$. If the vectors are always concentrated in T_y, the span of them will be contained in T_y and therefore have dimension $n-r-s$. On the other hand, if we do the same for the original un-bloated oracles, we argue (again using self-reducibility) that the vectors $\mathbf{u} - \mathbf{u}'$ will be random in S_y and hence with enough of them we span the whole larger space of dimension $n-r$. Looking at the dimension of the spanned space therefore distinguishes the original and bloated duals, which we already showed was impossible. Thus, the original algorithm must at least *occasionally* output collisions that are not bad.

Step 3: Reducing to a Coset-Partition Function. We have now reduced our problem to proving the collision resistance of H when only given the oracles $\mathcal{P}, \mathcal{P}^{-1}$ but where the adversary does not have access to the oracle $\mathcal{D}$. We now prove such collision resistance, assuming the collision-resistance of some coset-partition function Q. This may seem counter-productive; after all, we deliberately moved away from the coset-partition structure of [AGKZ20]. Looking ahead, we will see that, since we have now stripped away the dual oracle, we actually *can* analyze the coset-partition structure.

We explain that given a coset-partition function Q, we can readily construct an instance of $\mathcal{P}, \mathcal{P}^{-1}$: $\mathcal{P}(x)$ sets $y = Q(x)$, and $\mathbf{u} = \left(x, 0^{k-n}\right)$. This also makes $\mathcal{P}^{-1}\left(y, \left(x, 0^{k-n}\right)\right)$ trivial as x is already present in the clear; we just need to verify that $Q(x) = y$ before outputting x (and outputting $\perp$ otherwise). This gives a valid instance of the oracles $\mathcal{P}, \mathcal{P}^{-1}$ since the pre-image sets of Q are already cosets, so outputting x padded with 0's satisfies the structure of $\mathcal{P}$. The oracles $\mathcal{P}, \mathcal{P}^{-1}$ clearly do not have the correct distribution, but we can once more apply the random self-reduction to simulate a proper random instance of $\mathcal{P}, \mathcal{P}^{-1}$.

Importantly, observe that the reduction which simulate $\mathcal{P}, \mathcal{P}^{-1}$ only require access to Q *in the forward direction*. This means we have reduced our problem to the collision resistance of Q, given only forward queries to Q.

Step 4: Constructing Hard Coset-Partition Functions. It may appear that we are still stuck: while we have eliminated the dual and inverse oracles, we have re-introduced the extra structure of a coset partition function. Thankfully, our reduction from the previous Step 3 actually shows that *any* coset partition function Q can be used to simulate the oracles $\mathcal{P}, \mathcal{P}^{-1}$, as long as $Q : \{0,1\}^n \to \{0,1\}^r$ (and each preimage set, which is a coset, has size 2^{n-r}). This means that finding any distribution over such Q that is provably collision resistant, will constitute that our construction is collision resistant.

We now explain a simple construction of such a function Q. We start by observing that a 2-to-1 function is trivially a coset partition function. This is because any pre-image set, which is 2 points x, x', is automatically a coset of the 1-dimensional subspace $S := \{0^n, x \oplus x'\} \subseteq \mathbb{Z}_2^n$ with a constant shift of x (or equivalently, a constant shift of x'). Also, the known quantum collision

lower-bounds [AS04, Zha15] imply that a random 2-to-1 function is provably collision resistant. Now, just any random 2-to-1 function is not enough for us, but the following can be done.

We observe that a straightforward combination of several existing results proves that a random 2-to-1 function, with the added property that it is shrinking by 1 bit, is also collision resistant. Given such shrinking ℓ i.i.d. random 2-to-1 functions $H_1 : \{0,1\}^{n'} \to \{0,1\}^{n'-1}, \cdots, H_\ell : \{0,1\}^{n'} \to \{0,1\}^{n'-1}$, take $Q : \{0,1\}^{n' \cdot \ell} \to \{0,1\}^{n' \cdot \ell - \ell}$ to be the ℓ-wise parallel application of them. The pre-image sets are then the direct sums of ℓ pre-image sets of the underlying 2-to-1 functions, and direct sums of cosets are cosets. Parallel application also preserves collision resistance. Putting everything together we set $n := n' \cdot \ell$, $r := n' \cdot \ell - \ell$, which proves the oracle-security of our construction, thereby proving Theorems 1 and 2.

2.3 Obfuscating PRPs (Section 5)

Our next goal is to turn our oracle proof into a standard-model proof. The natural approach is, instead of providing oracles for the various functions $\mathcal{P}, \mathcal{P}^{-1}, \mathcal{D}$, to provide obfuscations of these functions. To make the funcitons efficient, we will replace the random permutation Π with a pseudo-random permutation, and the random choices of $\mathbf{A}_y, \mathbf{b}_y$ with values generated by a pseudorandom function.

But we immediately run into a problem: there are simply no known techniques for proving the security of obfuscated pseudorandom permutations when using the standard notion of indistinguishability obfuscation (iO). This is because iO only provides a seemingly very weak guarantee: that functionally-equivalent programs are indistinguishable. In order to use iO, some of the program transformations actually need happen outside of the iO, which in turn requires other cryptographic techniques. Most of the iO literature follows the punctured programming approach [SW14], which uses the notion of a "punctured PRF" [KPTZ13, BW13, BGI14]. These, very roughly, allow for giving out a program that either computes the PRF correctly, *or* the program computes the PRF everywhere but a single (known) point, and for that one point the output is uniformly random. Security says that these two programs are indistinguishable.

Permutable PRPs. Unfortunately, pseudorandom permutations provably cannot be punctured in the same sense as PRFs, as explained by [BKW17]. Roughly, the reason is that replacing the output at a single point with a uniform random value means the function is no longer a permutation.

We instead define the notion of a *permutable* PRP. Here, given k a PRP secret key and some fixed permutation Γ, it is possible to produce a circuit which computes either (1) $\Pi(k, \cdot)$ and its inverse correctly, or (2) $\Gamma(\Pi(k, \cdot))$ and its inverse. Security requires that (1) and (2) are indistinguishable, even if Γ is known. This avoids the impossibility of [BKW17] since we always maintain that the program being computed is a permutation. When used in iO proofs, permutable PRPs readily give that $\mathsf{iO}(\Pi(k, \cdot))$ is computationally indistinguishable from $\mathsf{iO}(\Gamma(\Pi(k, \cdot)))$. This even holds true if given obfuscated programs for the inverses, and even for more complicated programs that may query the permutations several times. Observe that a similar statement for puncturable pseudorandom functions readily follows from [CLTV15], but that case inherently has no inverse.

Constructing Permutable PRPs for Neighbor Swaps. For now we will focus on an extremely simple setting, where we restrict Γ to be a swap between z and $z+1$ for some value z, but leave all other points unaffected. We will call such Γ a *neighbor swap*.

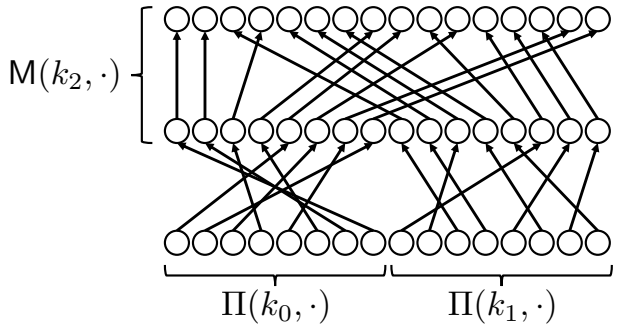

Figure 1: The PRP construction $\Pi(k, \cdot)$ of [GP07]. There are 2^n input nodes on the bottom, one per input, and 2^n output nodes. The left and right halves correspond to inputs starting with 0 and 1 rexpectively. Here, k_0, k_1, k_2 are keys pseudorandomly derived from k. Then $\mathsf{M}(k_2, \cdot)$ is a pseudorandom object we call a Merge which preserves the order of each half but otherwise pseudorandomly scrambles them. $\Pi(k_0, \cdot)$ and $\Pi(k_1, \cdot)$ are then recursive calls to the construction on inputs of length $n - 1$.

Even for the simple case of a neighbor swap, a permutable PRP is non-trivial. An "obvious" choice is to take a PRP built from a pseudorandom *function* (PRF), and instantiate the PRF with a puncturable PRF. But it is not at all clear a priori how puncturing the underlying PRF allows for swapping outputs.

In fact, this strategy *cannot* work in general. For example, we argue that it cannot work for Luby-Rackoff PRPs. To see this, we observe that in the setting where the domain size is polynomial, a neighbor-swap PRP actually is also a standard PRP. This is because the neighbor-swap property ensures that you can computationally undetectably swap z and $z+1$ in the output truth table. Since any permutation on a polynomial-domain can be decomposed into a polynomial-length sequence of neighbor swaps, this says that you can apply a *random* permutation un-detectably. But composing with a random permutation actually gives a truly random permutation, thus showing that the truth table of the PRP is computationally indistinguishable from the truth table of a random permutation. We can also scale this argument up, and show that if the neighbor-swap PRP is sub-exponentially secure, then it must be secure against a "truth-table" adversary that is provided the entire (exponential-sized) truth table. We then recall that standard PRP constructions such as Luby-Rackoff *cannot* achieve security in the small-domain/ truth-table setting [Pat01].

Guided by the above discussion, we look to the literature on small-domain permutations, specifically the work of [GP07]. While [GP07] present their construction in a very procedural way involving "permutators", "splitters" and "repartitors", we will present the idea in a more conceptual way that will help us explain our permuting algorithm. The first observation is that one can construct a random permutation as follows: divide into two equal sized piles (those strings starting with 0 and 1, respectively). Recursively randomly permute each pile independently. Then randomly merge the two piles together; this step preserves the order in each pile (which were already shuffled, so this is fine), but randomly determines how the two piles are interleaved. It is not hard to see that every permutation uniquely corresponds to a triple containing a merge and two recursive permutations. Thus this process perfectly simulates a random permutation. See Figure 1.

It turns out that permutations structured in this way can be evaluated efficiently. To evaluate on a point, we only need to evaluate *one* of the recursive permutations for the pile it belongs to,

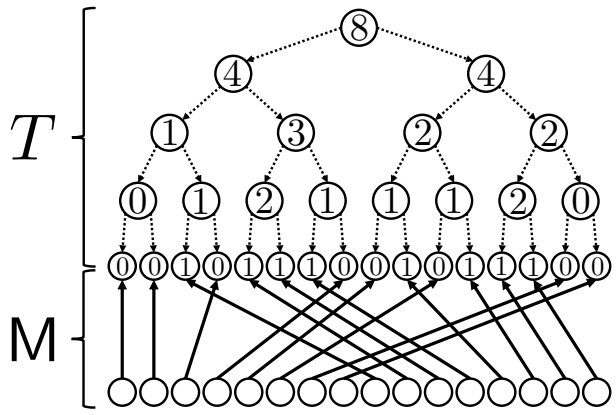

Figure 2: A merge M and the associated tally tree T, for the case $N_0 = N_1 = 8$, $N = 16$.

and then the merge operation (which we simply call a Merge). As there will be n levels of the recursion and each level only makes a single recursive call, this means that we can evaluate the overall permutation using n Merge evaluations. Then the Merge can be computed efficiently using a random data-structure we call a tally tree. A tally tree is a full binary tree on 2^n nodes, where leaf z corresponds to an *output* of the Merge, and is given the value of the first bit of the pre-image z. Then the internal nodes count the total number of 1's in the leaves of the sub-tree rooted at that node. An example tally tree T for a marge M is given in Figure 2.

Using a tally tree we can efficiently evaluate the Merge. We start by describing the inverse: since the Merge preserves the order within each half, we can compute the inverse of z just knowing (1) which half the pre-image of z belongs to, and (2) how many nodes to left of z go to the same half. This can be computed easily using the tally tree. Then the forward direction can be computed using a binary search algorithm which makes queries to the inverse and again exploits the order-preserving property.

The next observation is, rather than choosing a random merge and computing the tree in a bottom-up manner, we can actually sample the tree in a top-down manner. In this view, the value in each node follows an appropriate hypergeometric distribution based on the value of its parent. Then the random choices can be simulated using a pseudorandom function (PRF). The values in the tree are left implicitly determined by the PRF key, to be computed on the fly as needed during evaluation (which only visits a polynomial-sized portion of the tree since evaluation is efficient). The result is that keys can be small (independent of the domain size), and evaluation takes time poly-logarithmic in the domain size.

We start with this construction, and first show that we can permute z and $z+1$ by either permuting the Merge or by permuting one of the recursive calls to Π. In particular, if the pre-images of z and $z+1$ lie in different halves of the domain, then we can permute within the Merge. If the pre-images lie in the same half, then we cannot permute the Merge since this would violate the ordering of elements in that half. In this case, however, the order-preserving property of the Merge guarantees that the pre-images of $z, z+1$ are adjacent, and so we can instead permute within the recursive Π call for that half. See Figure 3.

Thus, we have reduced the task of permuting π to permuting the Merge, and we only need to concern ourselves with the case where the pre-images of $z, z+1$ lie in different halves. If we look at

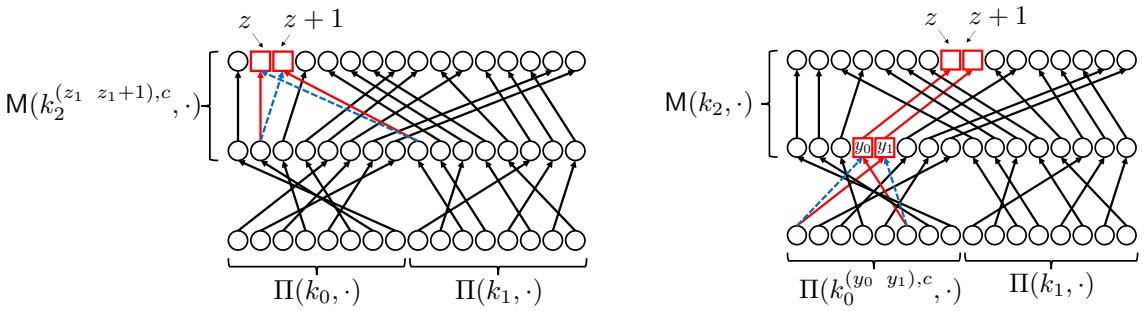

Figure 3: Our permuting algorithm. Here, $k^{(z\ z+1),c}$ means the permuted key, where $c = 0$ means no swapping occurs, and $c = 1$ means z and $z+1$ are swapped. Red squares indicate the points $z, z+1$, red solid arrows indicate the original permutation ($c = 0$), and blue dashed lines indicate the permuted permutation ($c = 1$). Let y_0, y_1 be the pre-images of $z, z+1$ in the Merge. Left: the case y_0, y_1 lie in different halves, where we permute $z, z+1$ by permuting the merge M. Right: the case that y_0, y_1 lie in the same half, where we permute $z, z+1$ by permuting the recursive application of Π, as indicated by the red squares.

the leaf nodes $z, z+1$ in the tally tree, since they have pre-images in different halves, this means one of the nodes has a 0 and one has a 1. Moreover, permuting $z, z+1$ corresponds to exchanging which is a 0 and which is a 1.

What we show is that such permuting can be accomplished by puncturing the underling pseudorandom function (PRF) that is used to generate the tally tree. We use a puncturable PRF that can be punctured at several points, resulting in those points being replaced with random values. Concretely, we puncture at the nodes $z, z+1$, as well as all nodes on the paths from these nodes to the root, and also the siblings of those nodes. By analyzing the induced hypergeometric distributions, we conclude after puncturing that the values at $z, z+1$ are actually statistically equally likely to be 0 or 1. Thus, we can swap their values without detection. See Figure 4 for an example.

Remark 11. *[GP07] is not particularly efficient from a practical perspective, due to having to sample from hypergeometric distributions. Other works have given more efficient constructions of small-domain PRPs [Mor05, HMR12, RY13, MR14], but it is unclear if these also give permutable PRPs when instantiated with a puncturable PRF.*

Extending to More General Permutations. Neighbor swaps are not sufficient for most applications. We therefore explain how to extend the construction to handle much more general permutations Γ.

The idea is simple, and builds upon the intuition that neighbor swaps are enough to generate all permutations. To get the circuits computing $\Pi(k,\cdot)$ or $\Gamma(\Pi(k,\cdot))$, we simply obfuscate those programs using iO. To show that the two cases are indistinguishable, we decompose Γ into a sequence of (exponentially-many) neighbor swaps, and perform a sequence of hybrids where we apply one neighbor-swap at a time.

There are two caveats to this approach. One is that we will need exponentially-many hybrids, and therefore we must rely on sub-expponentially-secure iO. The underlying puncturable PRF must

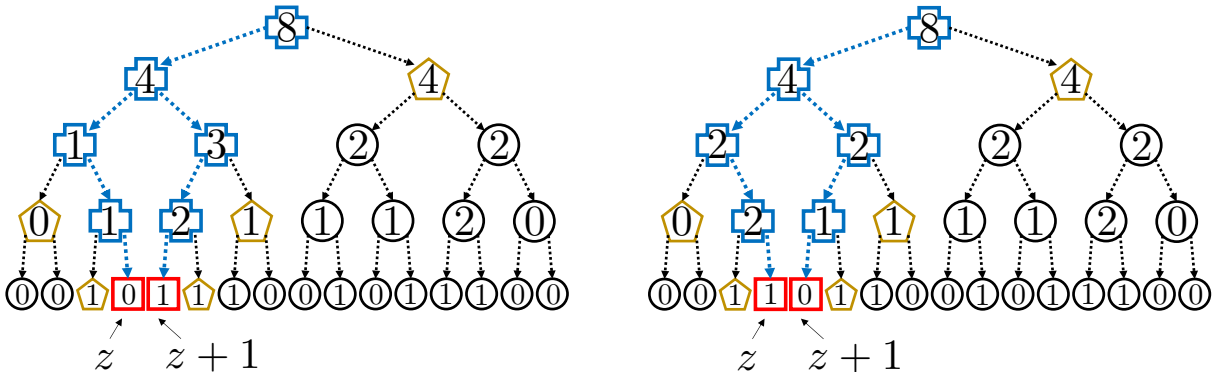

Figure 4: Left: The tally tree for a permuted merge key $k^{(z\ z+1),0}$, where no swapping happens. The red squares are the positions to be swapped $z, z+1$. The blue crosses are the paths from root to these nodes and the gold pentagons are the siblings of these paths; these nodes ahave all been punctured. Right: the tally tree for the permuted merge key $k^{(z\ z+1),1}$. This is identical to the left tree, except that we swapped the bits at z and $z+1$, and accordingly updated the tallies in the paths to $z, z+1$ (the blue crosses). The gold pentagons and all their descendants remain unchanged.

also be sub-exponentially secure, which is implied by sub-exponential one-way functions.

The more challenging caveat is the following. If we decompose Γ as $\Gamma = \tau_N \circ \tau_{N-1} \circ \cdots \circ \tau_1$ for neighbor-swaps τ_i, we actually need that each partial permutation $\Gamma_t := \tau_N \circ \tau_{N-1} \circ \cdots \circ \tau_t$ is computable by a small circuit. These need to be small because, at the appropriate hybrid, they will be hard-coded into the program that gets obfuscated. To maintain security given the obfuscated inverse circuits, we also need each Γ_t^{-1} to have small circuits. But in general, even if Γ, Γ^{-1} have small circuits, it is not obvious that Γ can be decomposed in a way which guarantees that each Γ_t, Γ_t^{-1} have smalls circuit. This leads to our Efficient Permutation Decomposition (EPD) Problem (Question 8), which asks whether such a guarantee is possible in general.[14]

We do not know how to resolve the EPD problem in general. Instead, we define a notion of "decomposable" circuits, which are Γ that can be decomposed into a (potentially exponentially-long) sequence in such a way that the Γ_t, Γ_t^{-1} all have small circuits.[15] Fortunately, decomposable circuits capture many natural types of permutations, such as general (non-neighbor) transpositions, linear cycles $j \mapsto j+1 \mapsto \cdots \mapsto \ell - 1 \mapsto \ell \mapsto j$, involutions (i.e., such that $\Gamma \circ \Gamma$ is the identity), and more. See Figure 5 for a longer list of examples of decomposable permutations, which are shown to be decomposable in Section 5. Our proof shows that an obfuscated neighbor-swap PRP is a permutable PRP for all decomposable circuits. This gives the informal Theorem 5, which is formalized in Section 5 as Theorem 51.

Remark 12. *Interestingly, despite showing that a wide range of permutations are decomposable, even for very simple permutations, decompositions can be tricky. For example, $x \mapsto 2x \bmod N$ for odd N is decomposable, though our proof uses the fact that the map is a merge with an efficiently computable tally tree, and shows that such tally trees allow for decomposition. This generalizes*

[14] Note that Question 8 was about decomposing into transpositions rather than neighbor swaps. But we will see that transpositions can be decomposed into neighbor swaps, showing that the two versions are equivalent.

[15] Note that there may be many possible decompositions into neighbor-swaps, and some decompositions may give small circuits while others may not.

Transpositions $i \leftrightarrow j$.
Linear cycles $j \to j+1 \to j+2 \to \cdots \to \ell \to j$.
Affine mod-2 permutations $\mathbf{x} \to \mathbf{A} \cdot \mathbf{x} + \mathbf{v} \bmod 2$ if $\det(A) \bmod 2 = 1$.
Products $(x, y) \to (\Gamma_0(x), \Gamma_1(x))$ for decomposable Γ_0, Γ_1.
Compositions $\Gamma = \Gamma_0 \circ \Gamma_1$ for decomposable Γ_0, Γ_1.
Controlled permutations $(x, y) \to (x, \Gamma_x(y))$ for decomposable Γ_x.
Efficient involutions ($\Gamma \circ \Gamma = \mathbf{I}$).
Conjugations $\Lambda^{-1} \circ \Gamma \circ \Lambda$ for decomposable Γ and efficient Λ, Λ^{-1}.
Permutations with ancillas $(x, 0^n) \to (\Gamma(x), 0^n)$ for efficient Γ, Γ^{-1}.

Figure 5: A non-exhaustive list of decomposable permutations. Note that for conjugations, Λ need not be decomposable. For ancillas, $\Gamma : \{0,1\}^n \to \{0,1\}^n$ need not be decomposable, but the permutation $(x, 0^n) \to (\Gamma(x), 0^n)$ is only a partial function that is not specified on the rest of the domain. The permutable PRP domain in this case is the entire space including ancillas.

to $x \mapsto ax \bmod N$ *for* polynomial a. *But the only way we know how to handle general linear maps requires decomposing* a *into the product* (mod N) *of polynomially-bounded scalars. This in turn requires that such polynomial scalars generate* $\mathbb{Z}_N^*$. *This is implied by the Extended Riemann Hypothesis [Bac90]. However, we leave a general unconditional decomposition, even for scalar multiplication, as an intriguing open question.*

Applications. We show that permutable PRPs can be very useful in obfuscation applications.[16] Concretely, we show that obfuscating a permutable PRP for general transpositions (which in particular are decomposable) immediately gives a *full-domain* trapdoor one-way permutation, yielding Theorem 6. The proof goes through several hybrids:

- Hybrid 0: The adversary is given an obfuscation of $\Pi(k, \cdot)$ and a challenge point y^*, and has to find x^* such that $y^* = \Pi(k, x^*)$.
- Hybrid 1: We choose a random output y', and "puncture" the program so if $\Pi(k, x) = y'$, the program outputs y^* instead of y'. Otherwise the program is unchanged. This means there are now two points x^*, x' that map to y^*. This change follows standard iO techniques.
- Hybrid 2: We use permuted key $k^{(x^* \ x'),0} \leftarrow \mathsf{Permute}\,(k, (x^* \ x'), 0)$ but where the transposition $(x^* \ x')$ is still turned off. Recall per Figure 5 that transpositions are decomposable. We use the permuted key instead of the real key k for all purposes of this hybrid. Since these are functionally equivalent, by the security of the iO, the transition to this hybrid is computationally undetectable.
- Hybrid 3: Now we compose the PRP with the transposition swapping x^*, x'. That is, we sample $k^{(x^* \ x'),1} \leftarrow \mathsf{Permute}\,(k, (x^* \ x'), 1)$ and use it instead of $k^{(x^* \ x'),0}$, which was used in the previous hybrid. By the security of the permutable PRP this change is computationally

[16] As we describe it, our applications will obfuscate a permutable PRP, where our permutable PRPs are built using obfuscation. We can equivalently simply obfuscate the underlying neighbor swap PRP, and all the hybrids in the proof of security for the permutable PRP can instead be carried out at the level of the application. However, for conceptual simplicity, we believe it is beneficial to describe our results using the permutable PRP abstraction.

indistinguishable. Note that, importantly we transpose x^* and x' both within the obfuscated program and also to verify the adversary's output (where we check that the circuit's output equals y^*). This swap means that now the adversary has to find x' instead of x^*. That is, after we move to this hybrid, we can make another change to how we check the success of this hybrid, take the original PRP key k and simply check that $y' = \Pi(k, x)$.

- Hybrid 4: Now we move to using the original key k inside the obfuscated circuit instead of the permuted key $k^{(x^* \; x'),1}$. Note that the functionality of the outside circuit (which is also obfuscated by iO) is unchanged because it is still the case that on both, x^*, x' it outputs y^*. This change is computationally indistinguishable by the security of the iO.

Finalizing the reduction. In the setting of the last hybrid experiment, we sample a random PRP key k and a uniformly random y'. We then obfuscate the circuit P_1, which computes the PRP with key k, checks if the output was y', and if so outputs y^* instead of y' (or in other words, does something unrelated to the original problem). The adversary then finds x' which eventually leads to finding y'. By standard iO techniques, finding such uniformly random y' is computationally intractable, even given the key k, the value y^* and the obfuscation of the program P_1.

An elaborated formal proof is found in Section 5 (Theorem 42) in the body of the paper.

2.4 OSS In the Standard Model (Section 6)

We finally turn to constructing OSS in the standard model. The idea is to simply replace the random permutation Π with a permutable PRP $\Pi(k, \cdot)$ and the have the matrix/vector pairs $\mathbf{A}_y, \mathbf{b}_y$ be generated pseudorandomly from a (puncturable) pseudorandom function. Fortunately, the preceding sections have already set up most of the ideas we need for the proof, as our sequence of reductions proving our oracle construction actually have natural cryptographic analogs.

For our random self-reduction, we need to argue that the simulated obfuscated program is indistinguishable from the real distribution. In the oracle case, we have perfect indistinguishability. In the standard-model case, we no longer have perfect indistinguishability since the simulated permutation Π is now the composition of two permutable PRPs, and the matrix/vector pairs $\mathbf{A}_y, \mathbf{b}_y$ are derived from underlying terms $\overline{\mathbf{A}}_y, \overline{\mathbf{b}}_y, \mathbf{C}_y, \mathbf{d}_y$, each of which are pseudorandomly generated. Fortunately, a standard hybrid over every y utilizing punctured PRF security lets us move to the correct distribution over $\mathbf{A}_y, \mathbf{b}_y$, and permutable PRP security lets us replace the composition of two permutable PRPs with a single PRP.

For dual-bloating, we follow the approach of [Zha19], which shows how to increase the size of of obfuscated subspaces using iO and one-way functions. We can likewise eliminate the dual as in the oracle case, as our anti-concentration argument works just as well in the standard-model. Finally, we can embed a coset-partition function just as we did in the oracle case, but now using our computational random self-reduction.

There are, however, a few important caveats to getting everything to work. The first is that, in order to show that the obfuscation of the composition of two permutable PRPs is indistinguishable from a single permutable PRP, we need our permutable PRPs to be decomposable. A per Theorem 39 we construct not only permutable PRPs but ones that are also decomposable in and of themselves. This allows for overall composition of permutable PRPs.

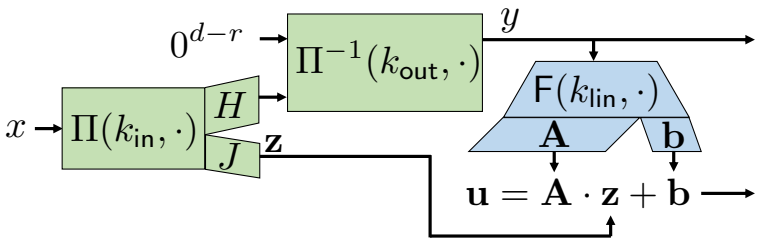

Figure 6: The program P that gets obfuscated to give the program $\mathcal{P}$. The programs P^{-1}, D are defined accordingly. Here, Π, Π^{-1} is a permutable PRP, and F is a puncturable pseudorandom function. The output of H is r bits, $\mathbf{z}$ is $n - r$ bits, and y is d bits.

Another caveat is that we need a post-quantum collision-resistant 2-to-1 function Q, since we can no longer rely on the unconditional existence relative to an oracle. It turns out that we also need this Q to have some extra properties, that were not present in the oracle case. Specifically, when we showed that embedding Q results in a valid instance of the oracles $\mathcal{P}, \mathcal{P}^{-1}$, we actually argued that this means there *exists* an implicit permutation Π determined by Q. Very roughly, since Q has collisions it therefore loses information, and Π outputs Q plus some additional bits to recover the lost information. For the oracle proof, only needed the existence of Π to argue that the simulated oracles had the same distribution as the real oracles. But for our standard-model argument, we will actually have a hybrid where Π is hard-coded into the obfuscated program. That means we need Π to have a small circuit. Unfortunately, Π essentially requires inverting Q, and so in general is computationally inefficient. Note that the inefficient circuit is not needed in normal usage, but just in the proof.

As a result, we need a collision-resistant 2-to-1 function that has a trapdoor. Fortunately, we can construct such "trapdoor" 2-to-1 hash functions from lattices, following a similar approach as the claw-free trapdoor functions built from LWE [BCM+18].

However, focusing on the LWE-based hash functions leads into the third and fourth caveats. For the third: The range of the LWE-based hash function is actually larger than the domain. In contrast, for our oracle construction described above, the function Q needs to be surjective in order for filling in the lost information to actually yield a permutation. But this means the range of Q must be smaller than the domain since it is many-to-one. This also presents a problem for invoking permutable PRP security, since permutable PRPs allow for composing with *permutations*, but our LWE-based hash function is no longer a permutation.

Fortunately, we can expand the range by padding y with 0's, and composing the output of the construction with *another* permutable PRP. This leads to our ultimate construction, which we describe visually in Figure 6. We can also split up the evaluation of Q into multiple stages, where each state is a (decomposable) permutation which allows us to invoke permutable PRP security.

The fourth caveat is that the LWE-based hash function is really only 2-to-1 on an overwhelming fraction of the domain, but it is easy to devise points where the function is only 1-to-1 on those points. This breaks the straightforward iO proof: when we embed Q into the programs, the programs will actually not have the same functionality due to the 1-to-1 points. This prevents us from naively using iO, which requires functionally equivalent programs.

To account for this, we add a trigger at all the sparse "bad" 1-to-1 points, and if this trigger occurs, the program behaves differently. We could try to embed this trigger in the original con-

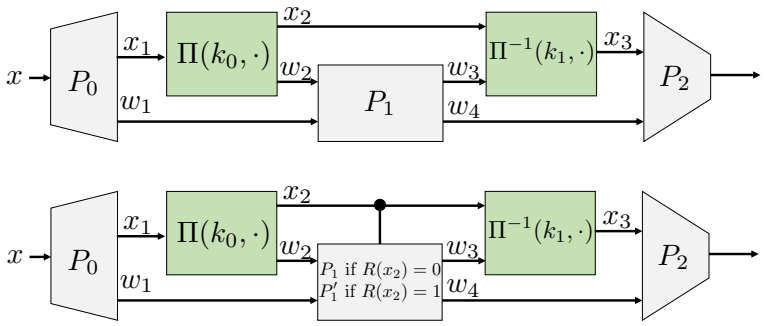

Figure 7: The original program P (top) and triggered program P' (bottom). Here, P_0, P_1, P_1', P_2 are arbitrary programs as long as they are independent of the keys k_0, k_1. Our indistinguishability result shows that for the case where $R(x_2)$ tests if x_2 (or some component thereof) is in a fixed sparse interval, then obfuscating these two programs yields computationally indistinguishable programs. Note that R is fixed and known, but due to the application of the permutable PRPs, the points that hit the trigger are computationally hidden once obfuscated.

struction, but it is not clear if we would be able to carry out all of our re-randomization steps; even if it could work, it would significantly complicate each of the steps. Instead, we add the trigger only in the proof. Standard iO techniques allow for adding sparse triggers on *random* points that are otherwise independent of the program, but not fixed triggers like we need.[17] Here, we use our permutable PRPs again. We show that, as long as the fixed trigger is sandwiched between two permutable PRPs (which they are, for our case, because we are already composing another PRP on the output) and meets some other technical conditions, we can un-detectably add the trigger. See Lemma 43 in Section 5 for a precise statement, and Figure 7 for a visual representation of the kinds of program changes which we can make.

Our proof of the sparse trigger works as follows. To trigger at a single *fixed* point y, we actually first add a trigger at a *random* point y'. Adding random triggers follows from standard iO techniques. Then, we use the permutable PRP property to exchange y and y', meaning now the trigger occurs at y. This step requires a permutable PRP both before and after the trigger, since after the trigger, we need to return y and y' to their original values. With a bit more work, we can extend this to interval triggers, as long as the interval is at most a sub-exponential fraction of the domain.

Once we add the trigger, then embedding Q actually yields an equivalent program and allows the proof to go through. By piecing all of the steps together, we obtain Theorems 3 and 4.

Remark 13. *One may wonder if there are other instantiations of the needed 2-to-1 collision-resistant hash function. Aside from needing the structure of being 2-to-1, another requirement that limits instantiations is the apparent need for a trapdoor. This seems to prevent us from using hash functions based on LPN [BLVW19, YZW+19] or super-singular isogeny graphs [CLG09]. One can always make a some-what tautological assumption that obfuscating the function which discards the last bit of a (permutable) PRP gives such a function (the trapdoor being the un-obfuscated key), though we do not know how to prove this based on any standard assumption. Another possibility is to look at constructions from group actions such as [AMR22], which can plausibly be post-quantum*

[17] The "bad" points are fixed once the hash function is chosen.

instantiated using ordinary *isogenies over elliptic curves. Their construction has a trapdoor and can be split up into decomposable parts analogous to the LWE-based construction, and has* most *of the domain being 2-to-1. However, a problem is that a non-negligible fraction of the domain is 1-to-1. This breaks the step where we embed a trigger into the obfuscated program, since the trigger is no longer sparse. An interesting open question is whether their construction can be modified so that an exponentially-small fraction of the domain is 1-to-1.*

3 Cryptographic Tools

We present the general cryptographic tools and techniques which are used in this work. Some of the results in this section are new and may be of independent interest.

Cryptographic Preliminaries. We use the following known primitives and notions. Both are implicitly classical primitives with security holding against quantum algorithms.

Definition 14 (Puncturable PRFs)**.** *A puncturable pseudorandom function (P-PRF) is a pair of efficient algorithms* $(\mathsf{F}, \mathsf{Punc}, \mathsf{Eval})$ *with associated output-length function* $m(\lambda)$ *such that:*

- $\mathsf{F} : \{0,1\}^\lambda \times \{0,1\}^* \to \{0,1\}^{m(\lambda)}$ *is a deterministic polynomial-time algorithm.*
- $\mathsf{Punc}(k, S)$ *is a probabilistic polynomial-time algorithm which takes as input a key* $k \in \{0,1\}^\lambda$ *and a set of points* $S \subseteq \{0,1\}^*$*. It outputs a punctured key* k^S*.*
- $\mathsf{Eval}\left(k^S, x\right)$ *is a deterministic polynomial-time algorithm.*
- **Correctness:** *For any* $\lambda \in \mathbb{N}$*,* $S \subseteq \{0,1\}^*$*,* $k \in \{0,1\}^\lambda$*,* $x \notin S$*, and* k^S *in the support of* $\mathsf{Punc}(k, S)$*, we have that* $\mathsf{Eval}\left(k^S, x\right) = \mathsf{F}(k, x)$*.*
- **Security:** *For any quantum polynomial-time algorithm* $\mathcal{A}$*, there exists a negligible function* ϵ *such that the following experiment with* $\mathcal{A}$ *outputs 1 with probability at most* $\frac{1}{2} + \epsilon(\lambda)$*:*
  - $\mathcal{A}(1^\lambda)$ *produces a set* $S \subseteq \{0,1\}^*$*.*
  - *The experiment chooses a random* $k \leftarrow \{0,1\}^\lambda$ *and computes* $k^S \leftarrow \mathsf{Punc}(k, S)$*. For each* $x \in S$*, it also sets* $y_x^0 := \mathsf{F}(k, x)$ *and samples* $y_x^1 \leftarrow \{0,1\}^{m(\lambda)}$ *uniformly at random. Then it chooses a random bit* b*. It finally gives* $k^S, \{(x, y_x^b)\}_{x \in S}$ *to* $\mathcal{A}$*.*
  - $\mathcal{A}$ *outputs a guess* b' *for* b*. The experiment outputs 1 if* $b' = b$*.*

Different security levels. For arbitrary functions $f_0, f_1 : \mathbb{N} \to \mathbb{N}$, we say that the P-PRF is $\left(f_0, \frac{1}{f_1}\right)$-secure if in the above the security part of the definition, we ask that the indistinguishability holds for every adversary of size $\leq f_0(\lambda)$ and we swap $\epsilon(\lambda)$ with $\frac{1}{f_1(\lambda)}$. Concretely, a sub-exponentially secure P-PRF scheme would be one such that there exists a positive real constant $c > 0$ such that the scheme is $\left(2^{\lambda^c}, \frac{1}{2^{\lambda^c}}\right)$-secure.

Definition 15 (Indistinguishability Obfuscation (iO))**.** *An indistinguishability obfuscator (iO) for Boolean circuits is a probabilistic polynomial-time algorithm* $\mathsf{iO}(\cdot, \cdot, \cdot)$ *with the following properties:*

- **Correctness:** *For all $\lambda, s \in \mathbb{N}$, Boolean circuits C of size at most s, and all inputs x to C,*
$$\Pr\left[\mathsf{O}_C(x) = C(x) \ : \ \mathsf{O}_C \leftarrow \mathsf{iO}\left(1^\lambda, 1^s, C\right)\right] = 1 \ .$$
- **Security:** *For every polynomial $\mathsf{poly}(\cdot)$ there exists a negligible function ϵ such that the following holds. Let $\lambda, s \in \mathbb{N}$, and let C_0, C_1 two classical circuits of (1) the same functionality (i.e., for every possible input they have the same output) and (2) both have size $\leq s$.*
$$\left\{\mathsf{O}_{C_0} \ : \ \mathsf{O}_{C_0} \leftarrow \mathsf{iO}\left(1^\lambda, 1^s, C_0\right)\right\}$$
$$\approx_{(\mathsf{poly}(\lambda),\epsilon(\lambda))} \left\{\mathsf{O}_{C_1} \ : \ \mathsf{O}_{C_1} \leftarrow \mathsf{iO}\left(1^\lambda, 1^s, C_1\right)\right\} \ .$$

Different security levels. For arbitrary functions $f_0, f_1 : \mathbb{N} \to \mathbb{N}$, we say that an iO scheme is $\left(f_0, \frac{1}{f_1}\right)$-secure if in the above the security part of the definition, we swap poly with a concrete function f_0 and the negligible function with the concrete $\frac{1}{f_1}$. Concretely, a sub-exponentially secure iO scheme would be one such that there exists a positive real constant $c > 0$ such that the scheme is $\left(2^{\lambda^c}, \frac{1}{2^{\lambda^c}}\right)$-secure.

Definition 16 (Lossy Functions)**.** *A lossy function (LF) scheme consists of classical algorithms* $(\mathsf{LF.KeyGen}, \mathsf{LF.F})$ *with the following syntax.*

- $\mathsf{pk} \leftarrow \mathsf{LF.KeyGen}\left(1^\lambda, b, 1^\ell\right)$*: a probabilistic polynomial-time algorithm that gets as input the security parameter $\lambda \in \mathbb{N}$, a bit b and a lossyness parameter $\ell \in \mathbb{N}$, $\lambda \geq \ell$. The algorithm outputs a public key.*
- $y \leftarrow \mathsf{LF.F}\left(\mathsf{pk}, x\right)$*: a deterministic polynomial-time algorithm that gets as input the security parameter $\lambda \in \mathbb{N}$, the public key pk and an input $x \in \{0,1\}^\lambda$ and outputs a string $y \in \{0,1\}^m$ for some $m \geq \lambda$.*

The scheme satisfies the following guarantees.

- **Statistical Correctness for Injective Mode:** *There exists a negligible function $\mathsf{negl}(\cdot)$ such that for every $\lambda, \ell \in \mathbb{N}$,*
$$\Pr_{\mathsf{pk}\leftarrow\mathsf{LF.KeyGen}\left(1^\lambda,0,1^\ell\right)}\left[\left|Img\left(\mathsf{LF.F}\left(\mathsf{pk},\cdot\right)\right)\right| = 2^\lambda\right] \geq 1 - \mathsf{negl}(\lambda) \ .$$
- **Statistical Correctness for Lossy Mode:** *There exists a negligible function $\mathsf{negl}(\cdot)$ such that for every $\lambda, \ell \in \mathbb{N}$,*
$$\Pr_{\mathsf{pk}\leftarrow\mathsf{LF.KeyGen}\left(1^\lambda,1,1^\ell\right)}\left[\left|Img\left(\mathsf{LF.F}\left(\mathsf{pk},\cdot\right)\right)\right| \leq 2^\ell\right] \geq 1 - \mathsf{negl}(\lambda) \ .$$
- **Security:** *For every polynomial $\mathsf{poly}(\cdot)$ there exists a negligible function ϵ such that the following holds. Let $\lambda, \ell \in \mathbb{N}$, then (note that in the following computational indistinguishability, the security parameter is ℓ and not λ),*
$$\left\{\mathsf{pk}_0 \ : \ \mathsf{pk}_0 \leftarrow \mathsf{LF.KeyGen}\left(1^\lambda, 0, \ell\right)\right\}$$
$$\approx_{(\mathsf{poly}(\ell),\epsilon(\ell))} \left\{\mathsf{pk}_1 \ : \ \mathsf{pk}_1 \leftarrow \mathsf{LF.KeyGen}\left(1^\lambda, 1, \ell\right)\right\} \ .$$

Different security levels. For arbitrary functions $f_0, f_1 : \mathbb{N} \to \mathbb{N}$, we say that an LF scheme is $\left(f_0, \frac{1}{f_1}\right)$-secure if in the above the security part of the definition, we swap poly with a concrete function f_0 and the negligible function with the concrete $\frac{1}{f_1}$. Concretely, a sub-exponentially secure LF scheme would be one such that there exists a positive real constant $c > 0$ such that the scheme is $\left(2^{\ell^c}, \frac{1}{2^{\ell^c}}\right)$-secure.

3.1 Two iO Techniques

Here, we recall two standard iO techniques that we will abstract as useful lemmas.

Sparse Random Triggers. Let P be some program and P' an arbitrary different program. Let R be a function with range $[N]$ for some N that is exponential in the security parameter. Let J_y (for 'join') be the program $J_y(x) = \begin{cases} P(x) \text{ if } R(x) \neq y \\ P'(x) \text{ if } R(x) = y \end{cases}$.

Lemma 17. *Suppose one-way functions exist. For sufficiently large polynomial s and for y chosen uniformly in $\{0,1\}^\lambda$, $\mathsf{iO}(1^\lambda, 1^s, P)$ and $\mathsf{iO}(1^\lambda, 1^s, J_y)$ are computationally indisitnguishable even given the description of P. Moreover, y is computationally unpredictable given $P, \mathsf{iO}(1^\lambda, 1^s, J_y)$*

Proof. We first prove indistinguishability through a sequence of hybrid programs:

- Hyb_0: The original obfuscation of P.

Here, the adversary is given $\mathsf{iO}(1^\lambda, 1^s, P)$.

- Hyb_1: Adding a uniformly random trigger that applies only if it is in the image of a sparse PRG.

We assume an injective length-doubling pseudorandom generator $\mathsf{PRG} : [N] \to [N]^2$. These follow from injective one-way functions, which in turn follow from plain one-way functions and iO [BPW16]. Here, we choose a random $w \leftarrow [N]^2$. The adversary is given $\mathsf{iO}(1^\lambda, 1^s, J'_w)$ where $J'_w(x) = \begin{cases} P(x) \text{ if } \mathsf{PRG}(R(x)) \neq w \\ P'(x) \text{ if } \mathsf{PRG}(R(x)) = w \end{cases}$. Note that since PRG is length-doubling, we have that with overwhelming probability over the choice of w, the second line of J'_w will never be triggered. Therefore, J'_w is functionally equivalent to P. Therefore, by iO security, as long as s is larger than the size of J'_w (which is larger than the size of P), hybrids 0 and 1 are indistinguishable.

- Hyb_2: Changing the trigger to be a random element inside the image of the PRG.

Here, we switch to $w = \mathsf{PRG}(y)$ for a random y. Indistinguishability from Hybrid 1 follows immediately from the pseudorandomness of PRG.

- Hyb_3: Dropping the use of the PRG and checking its image directly.

Now the adversary is given $\mathsf{iO}(1^\lambda, 1^s, J_y)$ for a random y. Observe that since $w = \mathsf{PRG}(y)$ and PRG is injective, J_y and J'_w have equivalent functionalities. Therefore, by iO security, hybrids 2 and 3 are indistinguishable. This completes the proof of indistinguishability.

For the computational unpredictability of y, consider an adversary starting in hybrid 3 which outputs y with probability ϵ. Since the indistinguishability of hybrids 2 and 3 did not rely on the randomness of y, we can switch to hybrid 2 and still obtain an adversary that outputs y with probability at least $\epsilon - \mathsf{negl}$. Now we observe that the view of the adversary only depends on $w = \mathsf{PRG}(y)$, and in the end the adversary produces y with non-negligible probability. Thus, by a straightforward reduction to the one-wayness of PRG, we conclude that $\epsilon - \mathsf{negl}$, and hence ϵ itself, must be negligible. □

Swapping distributions. We now move to the next standard technique. Let $\{D_0^x\}_x, \{D_1^x\}_x$ be two families of distributions over the same domain $\mathcal{Y}$, which can also be thought of as deterministic functions $D_0(x;r)$, $D_1(x;r)$ that take as input an index x and some random coins r. Let P be a program that makes queries to an oracle $O : \mathcal{X} \to \mathcal{Y}$ for some set $\mathcal{X}$. Then we have the following:

Lemma 18. *Let $(\mathsf{F}, \mathsf{Punc})$ be a $(f_{\mathsf{F}}, \delta_{\mathsf{F}})$-secure puncturable PRF and iO be a $(f_{\mathsf{iO}}, \delta_{\mathsf{iO}})$-secure iO. Let $\mathcal{X}$ a finite set and let $D_0 := \{D_{0,x}\}_{x\in\mathcal{X}}$,$D_1 := \{D_{1,x}\}_{x\in\mathcal{X}}$ two ensembles of distributions, such that for every $x \in \mathcal{X}$, $D_{0,x}$, $D_{1,x}$ are (f_D, δ_D)-indistinguishable. Let $E_0(x) = D_0(x; \mathsf{F}(k,x))$ and $E_1(x) = D_1\left(x; \mathsf{F}\left(\overline{k}, x\right)\right)$. Then for a sufficiently large polynomial s, $\mathsf{iO}\left(1^\lambda, 1^s, P^{E_0}\right)$ and $\mathsf{iO}\left(1^\lambda, 1^s, P^{E_1}\right)$ are $(\min\left(f_{\mathsf{F}}, f_{\mathsf{iO}}, f_D\right), O\left(|\mathcal{X}| \cdot (\delta_{\mathsf{F}} + \delta_{\mathsf{iO}} + \delta_D)\right))$-computationally indistinguishable, where $k, \overline{k} \leftarrow \{0,1\}^\lambda$ are uniformly random keys.*

Proof. We assume $\mathcal{X} = [N]$ by giving some ordering to the set. We prove security through a sequence of hybrids.

$\mathsf{Hyb}_{i.0}$: The adversary gets $\mathsf{iO}(1^\lambda, 1^s, P^{E_{i,0}})$, where $E_{i,0}$ has $k, \overline{k}$ hard-coded and is defined as $E_{i,0}(x) = \begin{cases} D_0(x; \mathsf{F}(k,x)) & \text{if } x \geq i \\ D_1(\mathsf{F}(x; \overline{k}, x)) & \text{if } x < i \end{cases}$.

$\mathsf{Hyb}_{i.1}$: The adversary gets $\mathsf{iO}(1^\lambda, 1^s, P^{E_{i,1}})$, where to generate $E_{i,1}$, we compute $k^i \leftarrow \mathsf{Punc}(k,i), \overline{k}^i \leftarrow \mathsf{Punc}(\overline{k}, i)$, sample $y \leftarrow D_0(i; \mathsf{F}(k,i))$ and let $E_{i,1}(x) = \begin{cases} D_0(x; \mathsf{F}(k,x)) & \text{if } x > i \\ y & \text{if } x = i \\ D_1(x; \mathsf{F}(\overline{k}, x)) & \text{if } x < i \end{cases}$. Observe that by our choice of y, $E_{i,1}$ is identical to $E_{i,0}$, and hence the programs $P^{E_{i,0}}$ and $P^{E_{i,1}}$ are equivalent. Thus, by the security of iO, Hybrids $i.0$ and $i.1$ are indistinguishable except with probability δ_{iO}.

$\mathsf{Hyb}_{i.2}$: Here, we still obfuscate $P^{E_{i,1}}$, but instead switch to $y \leftarrow D_0(i; r)$ for fresh random coins r. Observe that the entire experiment except for y is simulatable using just the punctured key k^i, and the only difference for y is that we replace $\mathsf{F}(k,i)$ with a random string. Thus Hybrids $i.1$ and $i.2$ are indistinguishable except with probability δ_{F}.

$\mathsf{Hyb}_{i.3}$: Now we change to $y \leftarrow D_1(r)$. Hybrids $i.2$ and $i.3$ are indistinguishable except with probability ϵ.

$\mathsf{Hyb}_{i.4}$: Now we change to $y \leftarrow D_1(\mathsf{F}(\overline{k}, i))$. Hybrids $i.3$ and $i.4$ are indistinguishable except with probability δ_{F}.

Next, we observe that $E_{i,1}$, when using $y \leftarrow D_1(i; \mathsf{F}(\overline{k}, i))$, is actually functionally equivalent to $E_{i+1,0}$. Thus, we see that the programs $P^{E_{i,1}}$ and $P^{E_{i+1,0}}$ are functionally equivalent. By iO security, we therefore have that Hybrid $i.4$ and $(i+1).0$ are indistinguishable except with probability δ_{iO}.

The proof then follows by observing that Hybrid 1.0 corresponds to $\mathsf{iO}(1^\lambda, 1^s, P^{D_0(\mathsf{F}(k,\cdot))})$ and Hybrid $(N+1).0$ corresponds to $\mathsf{iO}(1^\lambda, 1^s, P^{D_1(\mathsf{F}(\overline{k},\cdot))})$

□

3.2 Information-Theoretical Hardness of Hidden Subspace Detection

One of our central objects in this paper are quantumly accessible classical functions that check membership in some secret linear subspace $S \subseteq \mathbb{Z}_2^k$.

Information-Theoretical Subspace Hiding. We start with a quantum lower bound for detecting a change between two oracles: One allows access to membership check for some given (known) subspace S, and the other allows access to membership check in a random superspace T of S. This is an information-theoretical version of the subspace-hiding obfuscation introduced in [Zha19].

Lemma 19. *Let $k, r, s \in \mathbb{N}$ such that $r + s \leq k$ and let $S \subseteq \mathbb{Z}_2^k$ a subspace of dimension r. Let $\mathcal{S}_s$ the uniform distribution over subspaces T of dimension $r + s$ such that $S \subseteq T \subseteq \mathbb{Z}_2^k$. For any subspace $S' \subseteq \mathbb{Z}_2^k$ let $\mathcal{O}_{S'}$ the oracle that checks membership in S' (outputs 1 if and only if the input is inside S').*

Then, for every oracle-aided quantum algorithm $\mathcal{A}$ making at most q quantum queries, we have the following indistinguishability over oracle distributions.

$$\{\mathcal{O}_S\} \approx_{O\left(\frac{q\cdot s}{\sqrt{2^{k-r-s}}}\right)} \{\mathcal{O}_T \ : \ T \leftarrow \mathcal{S}_s\} \ .$$

Proof. We prove the claim by a hybrid argument, increasing the dimension of the random superspace T by 1 in each step, until we made an increase of s dimensions. In the first step we consider a matrix $\mathbf{B} \in \mathbb{Z}_2^{k\times(k-r)}$, the columns of which form a basis for $S^\perp$. Note that the oracle $\mathcal{O}_S$ can be described as accepting $\mathbf{x} \in \mathbb{Z}_2^k$ iff $\mathbf{x}^T \cdot \mathbf{B} = 0^{k-r}$.

Next we sample a uniformly random $\mathbf{a} \in \mathbb{Z}_2^{k-r}$ and consider the oracle $\mathcal{O}_{S_1}$ that accepts $\mathbf{x} \in \mathbb{Z}_2^k$ iff either $\mathbf{x}^T \cdot \mathbf{B} = 0^{k-r}$ or $\mathbf{x}^T \cdot \mathbf{B} = \mathbf{a}$. Two things can be verified: (1) Due to the randomness of $\mathbf{a}$, by standard quantum lower bounds, $\{\mathcal{O}_S\} \approx_{O\left(\frac{q}{\sqrt{2^{k-r}}}\right)} \{\mathcal{O}_{S_1} \ : \ \mathbf{a} \leftarrow \mathbb{Z}_2^{k-r}\}$, and (2) The set S_1 is a random superspace of S with dimension $r + 1$.

This proves our claim for $s = 1$. For a general s we can make an s-step hybrid argument, where at each step we have S_i which is a random $(r+i)$-dimensional superspace of S. Overall we get that for a q-query algorithm $\mathcal{A}$ the distinguishing advantage between $\{\mathcal{O}_S\}$ and $\{\mathcal{O}_{S_1} \ : \ T \leftarrow \mathcal{S}_s\}$ is

$$\sum_{i\in[s]} O\left(\frac{q}{\sqrt{2^{k-r-(i-1)}}}\right) \leq O\left(\frac{q\cdot s}{\sqrt{2^{k-r-s}}}\right) \ ,$$

as needed. □

We will also use the following corollary of Lemma 19. The corollary says that it is still hard to distinguish membership check between S and T, also when we duplicate the oracle access ℓ times. The corollary follows by a direct simulation reduction.

Corollary 20. *Let $k, r, s \in \mathbb{N}$ such that $r + s \leq k$ and let $S \subseteq \mathbb{Z}_2^k$ a subspace of dimension r. Let $\mathcal{S}_s$ the uniform distribution over subspaces T of dimension $r + s$ such that $S \subseteq T \subseteq \mathbb{Z}_2^k$. For any subspace $S' \subseteq \mathbb{Z}_2^k$ let $\mathcal{O}_{S'}$ the oracle that checks membership in S' (outputs 1 if and only if the input is inside S').*

Then, for every oracle-aided quantum algorithm $\mathcal{A}$ making at most q quantum queries, we have the following indistinguishability over oracle distributions.

$$\{\mathcal{O}_S^1, \cdots, \mathcal{O}_S^\ell\} \approx_{O\left(\frac{q\cdot\ell\cdot s}{\sqrt{2^{k-r-s}}}\right)} \{\mathcal{O}_T^1, \cdots, \mathcal{O}_T^\ell \ : \ T \leftarrow \mathcal{S}_s\} \ .$$

An additional corollary which follows by combining the above, with a standard hybrid argument on the first statement 19 is as follows. Note that in the below statement, the first oracle distribution is where each of the ℓ oralces samples an i.i.d. superspace T_i, and the second oracle distribution is where we sample T once, and then duplicate its oracle.

Corollary 21. *Let $k, r, s \in \mathbb{N}$ such that $r + s \leq k$ and let $S \subseteq \mathbb{Z}_2^k$ a subspace of dimension r. Let $\mathcal{S}_s$ the uniform distribution over subspaces T of dimension $r + s$ such that $S \subseteq T \subseteq \mathbb{Z}_2^k$. For any subspace $S' \subseteq \mathbb{Z}_2^k$ let $\mathcal{O}_{S'}$ the oracle that checks membership in S' (outputs 1 if and only if the input is inside S').*

Then, for every oracle-aided quantum algorithm $\mathcal{A}$ making at most q quantum queries, we have the following indistinguishability over oracle distributions.

$$\{\mathcal{O}_{T_1}, \cdots, \mathcal{O}_{T_\ell} \ : \ \forall i \in [\ell] : T_i \leftarrow \mathcal{S}_s\}$$

$$\approx_{O\left(\frac{q\cdot\ell\cdot s}{\sqrt{2^{k-r-s}}}\right)} \{\mathcal{O}_T^1, \cdots, \mathcal{O}_T^\ell \ : \ T \leftarrow \mathcal{S}_s\} \ .$$

The below is an information theoretical version of our Lemma 26 and corresponding proof.

Lemma 22. *Let $k, r, s \in \mathbb{N}$ such that $r + s \leq k$ and let $S \subseteq \mathbb{Z}_2^k$ a subspace of dimension r. Let $\mathcal{S}_s$ the uniform distribution over subspaces T of dimension $r + s$ such that $S \subseteq T \subseteq \mathbb{Z}_2^k$. For any subspace $S' \subseteq \mathbb{Z}_2^k$ let $\mathcal{O}_{S'}$ the oracle that checks membership in S' (outputs 1 if and only if the input is inside S').*

Assume there is an oracle-aided quantum algorithm $\mathcal{A}$ making at most q quantum queries and outputting a vector $\mathbf{u} \in \mathbb{Z}_2^k$ at the end of its execution, such that

$$\Pr\left[\mathcal{A}^{\mathcal{O}_T} \in \left(T^\perp \setminus \{0\}\right) \ : \ T \leftarrow \mathcal{S}_s\right] \geq \epsilon \ .$$

Also, denote $t := k - r - s$, $\ell := \frac{k(t+1)}{\epsilon}$ and assume (1) $\frac{t\cdot\frac{1}{\epsilon}}{2^{s-t}} \leq o(1)$ and (2) $\frac{q\cdot\ell^2\cdot s}{\sqrt{2^t}} \leq o(1)$. Then, it is necessarily the case that

$$\Pr\left[\mathcal{A}^{\mathcal{O}_T} \in \left(S^\perp \setminus T^\perp\right) \ : \ T \leftarrow \mathcal{S}_s\right] \geq \frac{\epsilon}{16 \cdot k \cdot (t+1)} \ .$$

Proof. We start with defining the following reduction $\mathcal{B}$, that will use the circuit $\mathcal{A}$ as part of its machinery.

The reduction $\mathcal{B}$. The input to $\mathcal{B}$ contains $\ell := \frac{k\cdot(t+1)}{\epsilon}$ samples of oracles $\left(\mathcal{O}^{(1)}, \cdots, \mathcal{O}^{(\ell)}\right)$, for $t := k - r - s$. Given the ℓ oracles, execute $\mathcal{A}^{\mathcal{O}^{(i)}}$ for every $i \in [\ell]$ and obtain ℓ vectors $\{u_1, \cdots, u_\ell\}$. Then, take only the vectors $\{v_1, \cdots, v_m\}$ that are inside $S^\perp$, and then compute the dimension of their span, $D := \dim\left(\mathsf{Span}\left(v_1, \cdots, v_m\right)\right)$. Note that the number of queries that $\mathcal{B}$ makes is $q \cdot \ell$.

Executing $\mathcal{B}$ on the oracle distribution $\mathcal{D}_1$. Consider the following distribution $\mathcal{D}_1$: Sample ℓ i.i.d superspaces $T_1, \cdots, T_\ell$, and for each of them, give access to its membership check oracle: $\mathcal{O}_{T_1}, \cdots, \mathcal{O}_{T_\ell}$. Let us see what happens when we execute $\mathcal{B}$ on a sample from the distribution $\mathcal{D}_1$.

Consider the ℓ vectors $\{u_1, \cdots, u_\ell\}$ obtained by executing $\mathcal{A}$ on each of the input oracles. Recall that $\ell := \frac{1}{\epsilon} \cdot k \cdot (t+1)$ and consider a partition of the vectors into $t+1$ consecutive sequences (or buckets), accordingly, each of length $\frac{1}{\epsilon} \cdot k$. In order to show that the probability for the reduction $\mathcal{B}$ to have $D \geq t+1$ is high, we show that with high probability, in each bucket $j \in [t+1]$ there is a vector u_i that's inside the corresponding dual $T_i^\perp$, but such that also the intersection between $T_i^\perp$ and each of the previous $j-1$ dual subspaces that were hit by $\mathcal{A}$, is only the zero vector 0^k. Note that the last condition indeed implies $D \geq t+1$.

For every $i \in [\ell]$ we define the probability p_i. We start with defining it for the indices in the first bucket, and then proceed to define it recursively for the rest of the buckets. For indices $i \in [\frac{1}{\epsilon} \cdot k]$ in the first bucket, p_i is the probability that given access to $\mathcal{O}_{T_i}$, the output of $\mathcal{A}$ is $u_i \in \left(T_i^\perp \setminus \{0\}\right)$, and in such case we define the i-th execution as successful. We denote by $T_{(1)}$ the first subspace in the first bucket where a successful execution happens (and define $T_{(1)} := \perp$ if no success happened). For any i inside any bucket $j \in ([t+1] \setminus \{1\})$ that is not the first bucket, we define p_i as the probability that (1) $u_i \in \left(T_i^\perp \setminus \{0\}\right)$ and also (2) the intersection between $T_i^\perp$ and each of the dual subspaces of the previous winning subspaces $T_{(1)}, \cdots, T_{(j-1)}$, is only $\{0^k\}$. That is, p_i is the probability that the output of the adversary hits the dual subspace, and also the dual does not have a non-trivial intersection with any of the previous successful duals. Similarly to the first bucket, we denote by $T_{(j)}$ the first subspace in bucket j with a successful execution.

We prove that with high probability, all $t+1$ buckets have at least one successful execution. To see this, we define the following probability p' which we show lower bounds p_i, and is defined as follows. First, let $\overline{T}_1, \cdots, \overline{T}_t$ any t subspaces, each of dimension $r+s$, thus the duals $\overline{T}_1^\perp, \cdots, \overline{T}_t^\perp$ are such that each has dimension t. $p'_{\left(\overline{T}_1, \cdots, \overline{T}_t\right)}$ is the probability that (1) when sampling $T^\perp$, the intersection of $T^\perp$ with each of the t dual subspaces $\overline{T}_1^\perp, \cdots, \overline{T}_t^\perp$ was only the zero vector, and also (2) the output of the adversary $\mathcal{A}$ was inside $T^\perp$. p' is defined is the minimal probability taken over all possible choices of t subspaces $\overline{T}_1, \cdots, \overline{T}_t$. After one verifies that indeed for every i we have $p' \leq p_i$, it is sufficient to lower bound p'.

Lower bound for the probability p'. The probability p' is for an event that's defined as the logical AND of two events, and as usual, equals the product between the probability p'_0 of the first event (the trivial intersection between the subspaces), times the conditional probability p'_1 of the second event (that $\mathcal{A}$ hits a non-zero vector in the dual $T^\perp$), conditioned on the first event.

First we lower bound the probability p'_0 by upper bounding the complement probability, that is, we show that the probability for a non-trivial intersection is small. Consider the random process of choosing a basis for a subspace T and note that it is equivalent to choosing a basis for the dual $T^\perp$. The process of choosing a basis for the dual has t steps, and in each step we choose a random vector in $S^\perp$ that's outside the span we aggregated so far. Given a dual subspace $\overline{T}^\perp$ of dimension t, what is the probability for the two subspaces to have only a trivial intersection? It is exactly the sum over $z \in [t]$ (which we think of as the steps for sampling $T^\perp$) of the following event: In the t-step process of choosing a basis for $T^\perp$, index z was the first to cause the subspaces to have a non-zero intersection. Recall that for each $z \in [t]$, the probability that z was such first index to cause an intersection, equals the probability that the z-th sampled basis vector for $T^\perp$ is a vector

that's inside the unified span of $\overline{T}^{\perp}$ and the aggregated span of $T^{\perp}$ so far, after $z-1$ samples. This amounts to the probability

$$\sum_{z\in[t]} \frac{|\overline{T}^{\perp}|\cdot 2^{z-1}}{|S^{\perp}|} = \sum_{z\in[t]} \frac{2^t\cdot 2^{z-1}}{2^{k-r}} = 2^{-s}\cdot \sum_{z\in\{0,1,\cdots,t-1\}} 2^z$$

$$= 2^{-s}\cdot\left(2^t-1\right) < 2^{t-s} \ .$$

Since the above is an upper bound on the probability for a non-trivial intersection between $T^{\perp}$ and one more single subspace, by union bound, the probability for $T^{\perp}$ to have a non-trivial intersection with at least one of the t subspaces $\overline{T}_1^{\perp}$, $\cdots$, $\overline{T}_t^{\perp}$ is upper bounded by $t\cdot 2^{t-s}$. This means that $p'_0 \geq 1-t\cdot 2^{t-s}$.

The lower bound for the conditional probability p'_1 is now quite easy: Note that since $\Pr[A|B] \geq \Pr[A]-\Pr[\neg B]$, letting A the event that $\mathcal{A}$ outputs a vector in the dual $T^{\perp}$ and B the event that $T^{\perp}$ has only a trivial intersection with all other t subspaces, we get $p'_1 \geq \epsilon - t\cdot 2^{t-s}$. By our assumption that $\frac{t\cdot\frac{1}{\epsilon}}{2^{s-t}} \leq o(1)$, we have $p'_1 \geq \frac{\epsilon}{2}$. Overall we got $p' := p'_0\cdot p'_1 \geq \left(1-t\cdot 2^{t-s}\right)\cdot\frac{\epsilon}{2} > \frac{\epsilon}{4}$.

Finally, to see why we get an overall high probability for $D\geq t+1$ on a sample from $\mathcal{D}_1$, observe the following. In each bucket there are $\frac{k}{\epsilon}$ attempts, each succeeds with probability at least $\frac{\epsilon}{4}$ and thus the overall success probability in a bucket is $\geq 1-e^{-\Omega(k)}$. Accordingly, the probability to succeed at least once in each of the $t+1$ buckets (and thus to satisfy $D\geq t+1$) is $\geq 1-(t+1)\cdot e^{-\Omega(k)}$, by considering the complement probability and applying union bound. Overall the probability for $D\geq t+1$ is thus $\geq 1-e^{-\Omega(k)}$.

Executing $\mathcal{B}$ on the distribution $\mathcal{D}_2$. Consider a different distribution $\mathcal{D}_2$: Sample T once, then allow an ℓ-oracle access to it, $\mathcal{O}_T^{(1)},\cdots,\mathcal{O}_T^{(\ell)}$. Note that $\mathcal{B}$ is a $q\cdot\ell$-query algorithm and thus by Corollary 21 there is the following indistinguishability of oracles with respect to $\mathcal{B}$:

$$\mathcal{D}_1 \approx_{O\left(\frac{q\cdot\ell^2\cdot s}{\sqrt{2^{k-r-s}}}\right)} \mathcal{D}_2 \ .$$

Since given a sample oracle from $\mathcal{D}_1$, the algorithm $\mathcal{B}$ outputs $D\geq t+1$ with probability $\geq 1-e^{-\Omega(k)}$, by the above indistinguishability, whenever we execute $\mathcal{B}$ on a sample from $\mathcal{D}_2$, then with probability at least $\geq 1-e^{-\Omega(k)}-O\left(\frac{q\cdot\ell^2\cdot s}{\sqrt{2^{k-r-s}}}\right) \geq 1-O\left(\frac{q\cdot\ell^2\cdot s}{\sqrt{2^{k-r-s}}}\right)$ we have $D\geq t+1$. By our assumption in the Lemma that $O\left(\frac{q\cdot\ell^2\cdot s}{\sqrt{2^{k-r-s}}}\right)\leq\frac{1}{2}$, with probability at least $\frac{1}{2}$ we have $D\geq t+1$ given a sample from $\mathcal{D}_2$. By an averaging argument, it follows that with probability at least $\frac{1}{2}\cdot\frac{1}{2}=\frac{1}{4}$ over sampling the superspace T, the probability p_T for the event where $D\geq t+1$, is at least $\frac{1}{2}\cdot\frac{1}{2}=\frac{1}{4}$. Let us call this set of superspaces T, "the good set" of samples, which by definition has fraction at least $\frac{1}{4}$. Recall two facts: (1) the dimension of $T^{\perp}$ is t, (2) The dimension D aggregates vectors inside $S^{\perp}$. The two facts together imply that in the event $D\geq t+1$, it is necessarily the case that there exists an execution index $i\in[\ell]$ in the reduction $\mathcal{B}$ where $\mathcal{A}$ outputs a vector in $\left(S^{\perp}\setminus T^{\perp}\right)$, given membership check in T.

For every T inside the good set we thus know that with probability $\frac{1}{4}$, one of the output vectors of $\mathcal{A}$ will be in $\left(S^{\perp}\setminus T^{\perp}\right)$. Since these are ℓ i.i.d. executions of $\mathcal{A}$, by union bound, for every T inside the good set, when we prepare an oracle access to T and execute $\mathcal{A}$, we will get $\mathcal{A}^{\mathcal{O}_T}\in\left(S^{\perp}\setminus T^{\perp}\right)$ with probability $\frac{1}{4\cdot\ell}$. We deduce that for a uniformly random T which we then prepare oracle access

to, the probability for $\mathcal{A}^{\mathcal{O}_T} \in \left(S^\perp \setminus T^\perp\right)$ is at least the probability for this event and also that T is inside the good set, which in turn is at least

$$\frac{1}{4} \cdot \frac{1}{4 \cdot \ell} = \frac{1}{16 \cdot \ell} := \frac{\epsilon}{16 \cdot k \cdot (t+1)} \ ,$$

which finishes our proof. □

3.3 Cryptographic Hardness of Hidden Subspace Detection

In this subsection we prove the cryptographic analogues of the information theoretical lower bounds from the previous section.

We start with stating the subspace-hiding obfuscation property of indistinguishability obfuscators from [Zha19].

Lemma 23. *Let $k, r, s \in \mathbb{N}$ such that $r + s \leq k$ and let $S \subseteq \mathbb{Z}_2^k$ a subspace of dimension r. Let $\mathcal{S}_s$ the uniform distribution over subspaces T of dimension $r + s$ such that $S \subseteq T \subseteq \mathbb{Z}_2^k$. For any subspace $S' \subseteq \mathbb{Z}_2^k$ let $C_{S'}$ some canonical classical circuit that checks membership in S', say be Gaussian elimination. Let iO an indistinguishability obfuscation scheme that is $(f(\lambda), \epsilon(\lambda))$-secure, and assume that $(f(\lambda), \epsilon(\lambda))$-secure injective one-way functions exist.*

Then, for every security parameter λ such that $\lambda \leq k - r - s$ and sufficiently large $p := p(\lambda)$ polynomial in the security parameter, we have the following indistinguishability,

$$\{\mathsf{O}_S \ : \ \mathsf{O}_S \leftarrow \mathsf{iO}\left(1^\lambda, 1^p, C_S\right)\} \approx_{(f(\lambda) - \mathsf{poly}(\lambda),\ s \cdot \epsilon(\lambda))}$$

$$\{\mathsf{O}_T \ : \ T \leftarrow \mathcal{S}_s, \mathsf{O}_T \leftarrow \mathsf{iO}\left(1^\lambda, 1^p, C_T\right)\} \ .$$

The following generalization of Lemma 23 is derived by a random self-reducibility argument and is formally proved by using an additional layer of indistinguishability obfuscation.

Lemma 24. *Let $k, r, s \in \mathbb{N}$ such that $r + s \leq k$ and let $S \subseteq \mathbb{Z}_2^k$ a subspace of dimension r. Let $\mathcal{S}_s$ the uniform distribution over subspaces T of dimension $r + s$ such that $S \subseteq T \subseteq \mathbb{Z}_2^k$. For any subspace $S' \subseteq \mathbb{Z}_2^k$ let $C_{S'}$ some canonical classical circuit that checks membership in S', say be Gaussian elimination. Let iO an indistinguishability obfuscation scheme that is $(f(\lambda), \epsilon(\lambda))$-secure, and assume that $(f(\lambda), \epsilon(\lambda))$-secure injective one-way functions exist.*

Then, for every security parameter λ such that $\lambda \leq k - r - s$ and sufficiently large $p := p(\lambda)$ polynomial in the security parameter, we have the following indistinguishability,

$$\{\mathsf{O}_S^1, \cdots, \mathsf{O}_S^\ell \ : \ \forall i \in [\ell], \mathsf{O}_S^i \leftarrow \mathsf{iO}\left(1^\lambda, 1^p, C_S\right)\} \approx_{(f(\lambda) - \ell \cdot \mathsf{poly}(\lambda),\ (2 \cdot \ell + s) \cdot \epsilon(\lambda))}$$

$$\{\mathsf{O}_T^1, \cdots, \mathsf{O}_T^\ell \ : \ T \leftarrow \mathcal{S}_s, \forall i \in [\ell], \mathsf{O}_T^i \leftarrow \mathsf{iO}\left(1^\lambda, 1^p, C_T\right)\} \ .$$

Proof. We first observe that as long as the circuit size parameter 1^p is sufficiently large, and specifically, larger than the size of an obfuscated version of the plain circuit C, then it is indistinguishable to tell whether said circuit is obfuscated under one or two layers of obfuscation. More precisely, due to the perfect correctness of obfuscation, the circuit C and an obfuscated $\mathsf{O}_C \leftarrow \mathsf{iO}\left(1^\lambda, 1^p, C\right)$ have the same functionality (for every sample out of the distribution of obfuscated versions of

C), and thus, as long as $p \geq |\mathsf{O}_C|$, then the distributions $\mathcal{D}_0 := \{\mathsf{O}_C \leftarrow \mathsf{iO}\left(1^\lambda, 1^p, C\right)\}$ and $\mathcal{D}_1 := \{\mathsf{O}_C^2 \leftarrow \mathsf{iO}\left(1^\lambda, 1^p, \mathcal{D}_0\right)\}$ are $(f(\lambda), \epsilon(\lambda))$-indistinguishable.

An implication of the above is that for a sufficiently large parameter p, the distribution

$$\{\mathsf{O}_S^1, \cdots, \mathsf{O}_S^\ell \ : \ \forall i \in [\ell], \mathsf{O}_S^i \leftarrow \mathsf{iO}\left(1^\lambda, 1^p, C_S\right)\}$$

is $(f(\lambda), \ell \cdot \epsilon(\lambda))$-indistinguishable from a distribution where C_S is swapped with an obfuscation of it (let us denote this modified distribution with $\mathcal{D}_S$), and the distribution

$$\{\mathsf{O}_T^1, \cdots, \mathsf{O}_T^\ell \ : \ T \leftarrow \mathcal{S}_S, \forall i \in [\ell], \mathsf{O}_T^i \leftarrow \mathsf{iO}\left(1^\lambda, 1^p, C_T\right)\}$$

is $(f(\lambda), \ell \cdot \epsilon(\lambda))$-indistinguishable from a distribution where C_T is swapped with an obfuscation of it (let us denote this modified distribution with $\mathcal{D}_T$). To complete our proof we will show that $\mathcal{D}_S$ and $\mathcal{D}_T$ are appropriately indistinguishable, and will get our proof by transitivity of computational distance.

The indistinguishability between $\mathcal{D}_S$ and $\mathcal{D}_T$ follows almost readily from the the subspace hiding Lemma 23: One can consider the reduction $\mathcal{B}$ that gets a sample O which is either from $\mathcal{D}_0 := \{\mathsf{O}_S \leftarrow \mathsf{iO}\left(1^\lambda, 1^p, C_S\right)\}$ or from $\mathcal{D}_1 := \{\mathsf{O}_T \leftarrow \mathsf{iO}\left(1^\lambda, 1^p, C_T\right)\}$ for an appropriately random superspace T of S. $\mathcal{B}$ then generates ℓ i.i.d obfuscations $\{\mathsf{O}^{(1)}, \cdots, \mathsf{O}^{(\ell)}\}$ of the (already obfuscated) circuit O and executes $\mathcal{A}$ on the ℓ obfuscations. One can see that when the input sample O for $\mathcal{B}$ came from $\mathcal{D}_0$ then the output sample of the reduction comes from the distribution $\mathcal{D}_S$, and when the input sample O for $\mathcal{B}$ came from $\mathcal{D}_1$ then the output sample of the reduction comes from the distribution $\mathcal{D}_T$. Since the reduction executes in complexity $\ell \cdot \mathsf{poly}(\lambda)$, this means that

$$\mathcal{D}_S \approx_{(f(\lambda) - \ell \cdot \mathsf{poly}(\lambda),\ s \cdot \epsilon(\lambda))} \mathcal{D}_T \ .$$

To conclude, by transitivity of computational indistinguishability, we get that the two distributions in our Lemma's statement are $(f(\lambda) - \ell \cdot \mathsf{poly}(\lambda),\ (2 \cdot \ell + s) \cdot \epsilon(\lambda))$-indistinguishable, as needed. □

A corollary which follows by combining the above Lemma 24, with a standard hybrid argument on the first lemma 23 is as follows.

Corollary 25. *Let $k, r, s \in \mathbb{N}$ such that $r + s \leq k$ and let $S \subseteq \mathbb{Z}_2^k$ a subspace of dimension r. Let $\mathcal{S}_s$ the uniform distribution over subspaces T of dimension $r + s$ such that $S \subseteq T \subseteq \mathbb{Z}_2^k$. For any subspace $S' \subseteq \mathbb{Z}_2^k$ let $C_{S'}$ some canonical classical circuit that checks membership in S', say be Gaussian elimination. Let iO an indistinguishability obfuscation scheme that is $(f(\lambda), \epsilon(\lambda))$-secure, and assume that $(f(\lambda), \epsilon(\lambda))$-secure injective one-way functions exist.*

Then, for every security parameter λ such that $\lambda \leq k - r - s$ and sufficiently large $p := p(\lambda)$ polynomial in the security parameter, we have the following indistinguishability,

$$\{\mathsf{O}_{T_1}, \cdots, \mathsf{O}_{T_\ell} \ : \ T_i \leftarrow \mathcal{S}_s \forall i \in [\ell], \mathsf{O}_{T_i} \leftarrow \mathsf{iO}\left(1^\lambda, 1^p, C_{T_i}\right)\}$$

$$\approx_{(f(\lambda) - \ell \cdot \mathsf{poly}(\lambda),\ (2 \cdot \ell \cdot s) \cdot \epsilon(\lambda))} \{\mathsf{O}_T^1, \cdots, \mathsf{O}_T^\ell \ : \ T \leftarrow \mathcal{S}_s, \forall i \in [\ell], \mathsf{O}_T^i \leftarrow \mathsf{iO}\left(1^\lambda, 1^p, C_T\right)\} \ .$$

Improved Results on Hardness of Concentration in Dual of Obfuscated Subspace. As part of this work we strengthen the main technical lemma (Lemma 5.1) from [Shm22b]. Roughly speaking, in [Shm22b] it is shown that an adversary $\mathcal{A}$ that gets an obfuscation O_T for a random superspace T of S, and manages to output a (non-zero) vector in the dual $T^\perp \setminus \{\mathbf{0}\}$ with probability ϵ, has to sometimes output vectors in $S^\perp \setminus T^\perp$, i.e. with probability at least $\Omega\left(\epsilon^2\right)/\mathsf{poly}(k)$. Below we strengthen the probability to $\Omega\left(\epsilon\right)/\mathsf{poly}(k)$.

Lemma 26 (IO Dual Subspace Anti-Concentration)**.** *Let $k, r, s \in \mathbb{N}$ such that $r + s \leq k$ and let $S \subseteq \mathbb{Z}_2^k$ a subspace of dimension r. Let $\mathcal{S}_s$ the uniform distribution over subspaces T of dimension $r+s$ such that $S \subseteq T \subseteq \mathbb{Z}_2^k$. For any subspace $S' \subseteq \mathbb{Z}_2^k$ let $C_{S'}$ some canonical classical circuit that checks membership in S', say be Gaussian elimination. Let iO an indistinguishability obfuscation scheme that is $\left(f(\lambda), \frac{1}{f(\lambda)}\right)$-secure, and assume that $\left(f(\lambda), \frac{1}{f(\lambda)}\right)$-secure injective one-way functions exist.*

Let $\lambda \in \mathbb{N}$ the security parameter such that $\lambda \leq k - r - s$ and let $p := p(\lambda)$ a sufficiently large polynomial in the security parameter. Denote by $\mathcal{O}_{\lambda,p,s}$ the distribution over obfuscated circuits that samples $T \leftarrow \mathcal{S}_s$ and then $\mathsf{O}_T \leftarrow \mathsf{iO}\left(1^\lambda, 1^p, C_T\right)$.

Assume there is a quantum algorithm $\mathcal{A}$ of complexity $T_{\mathcal{A}}$ such that,

$$\Pr\left[\mathcal{A}(\mathsf{O}_T) \in \left(T^\perp \setminus \{0\}\right) \ : \ \mathsf{O}_T \leftarrow \mathcal{O}_{\lambda,p,s}\right] \geq \epsilon \ .$$

Also, denote $t := k - r - s$, $\ell := \frac{k(t+1)}{\epsilon}$ and assume (1) $\frac{t \cdot \frac{1}{\epsilon}}{2^{s-t}} \leq o(1)$ and (2) $\frac{\ell \cdot \left(k^3 + \mathsf{poly}(\lambda) + T_{\mathcal{A}}\right)}{f(\lambda)} \leq o(1)$.

Then, it is necessarily the case that

$$\Pr\left[\mathcal{A}\left(\mathsf{O}_T\right) \in \left(S^\perp \setminus T^\perp\right) \ : \ \mathsf{O}_T \leftarrow \mathcal{O}_{\lambda,p,s}\right] \geq \frac{\epsilon}{16 \cdot k \cdot (t+1)} \ .$$

Proof. We start with defining the following reduction $\mathcal{B}$, that will use the circuit $\mathcal{A}$ as part of its machinery.

The reduction $\mathcal{B}$. The input to $\mathcal{B}$ contains $\ell := \frac{k \cdot (t+1)}{\epsilon}$ samples of obfuscations $\left(\mathsf{O}^{(1)}, \cdots, \mathsf{O}^{(\ell)}\right)$, for $t := k - r - s$. Given the ℓ obfuscations, execute $\mathcal{A}\left(\mathsf{O}^{(i)}\right)$ for every $i \in [\ell]$ and obtain ℓ vectors $\{u_1, \cdots, u_\ell\}$. Then, take only the vectors $\{v_1, \cdots, v_m\}$ that are inside $S^\perp$, and then compute the dimension of their span, $D := \dim\left(\mathsf{Span}\left(v_1, \cdots, v_m\right)\right)$. Note that the running time of $\mathcal{B}$ is $\ell \cdot T_{\mathcal{A}} + \ell \cdot k^3$, where $\ell \cdot T_{\mathcal{A}}$ is for producing the ℓ outputs of $\mathcal{A}$ and $\ell \cdot k^3$ is for (naively) executing Gaussian elimination ℓ times, to repeatedly check whether the new vector v_i adds a dimension i.e., whether it is outside of the span $\mathsf{Span}\left(v_1, \cdots, v_{i-1}\right)$ of the previous vectors.

Executing $\mathcal{B}$ on the distribution $\mathcal{D}_1$. Consider the following distribution $\mathcal{D}_1$: Sample ℓ i.i.d superspaces $T_1, \cdots, T_\ell$, and for each of them, send an obfuscation of it: $\mathsf{O}_{T_1}, \cdots, \mathsf{O}_{T_\ell}$. Let us see what happens when we execute $\mathcal{B}$ on a sample from the distribution $\mathcal{D}_1$.

Consider the ℓ vectors $\{u_1, \cdots, u_\ell\}$ obtained by executing $\mathcal{A}$ on each of the input obfuscations. Recall that $\ell := \frac{1}{\epsilon} \cdot k \cdot (t+1)$ and consider a partition of the vectors into $t+1$ consecutive sequences (or buckets), accordingly, each of length $\frac{1}{\epsilon} \cdot k$. In order to show that the probability for the reduction $\mathcal{B}$ to have $D \geq t+1$ is high, we show that with high probability, in each bucket $j \in [t+1]$ there is a vector u_i that's inside the corresponding dual $T_i^\perp$, but such that also the intersection between

$T_i^{\perp}$ and each of the previous $j-1$ dual subspaces that were hit by $\mathcal{A}$, is only the zero vector 0^k. Note that the last condition indeed implies $D \geq t+1$.

For every $i \in [\ell]$ we define the probability p_i. We start with defining it for the indices in the first bucket, and then proceed to define it recursively for the rest of the buckets. For indices $i \in [\frac{1}{\epsilon} \cdot k]$ in the first bucket, p_i is the probability that given O_{T_i}, the output of $\mathcal{A}$ is $u_i \in \left(T_i^{\perp} \setminus \{0\}\right)$, and in such case we define the i-th execution as successful. We denote by $T_{(1)}$ the first subspace in the first bucket where a successful execution happens (and define $T_{(1)} := \perp$ if no success happened). For any i inside any bucket $j \in ([t+1] \setminus \{1\})$ that is not the first bucket, we define p_i as the probability that (1) $u_i \in \left(T_i^{\perp} \setminus \{0\}\right)$ and also (2) the intersection between $T_i^{\perp}$ and each of the dual subspaces of the previous winning subspaces $T_{(1)}, \cdots, T_{(j-1)}$, is only $\{0^k\}$. That is, p_i is the probability that the output of the adversary hits the dual subspace, and also the dual does not have a non-trivial intersection with any of the previous successful duals. Similarly to the first bucket, we denote by $T_{(j)}$ the first subspace in bucket j with a successful execution.

We prove that with high probability, all $t+1$ buckets have at least one successful execution. To see this, we define the following probability p' which we show lower bounds p_i, and is defined as follows. First, let $\overline{T}_1, \cdots, \overline{T}_t$ any t subspaces, each of dimension $r+s$, thus the duals $\overline{T}_1^{\perp}, \cdots, \overline{T}_t^{\perp}$ are such that each has dimension t. $p'_{\left(\overline{T}_1, \cdots, \overline{T}_t\right)}$ is the probability that (1) when sampling $T^{\perp}$, the intersection of $T^{\perp}$ with each of the t dual subspaces $\overline{T}_1^{\perp}, \cdots, \overline{T}_t^{\perp}$ was only the zero vector, and also (2) the output of the adversary $\mathcal{A}$ was inside $T^{\perp}$. p' is defined is the minimal probability taken over all possible choices of t subspaces $\overline{T}_1, \cdots, \overline{T}_t$. After one verifies that indeed for every i we have $p' \leq p_i$, it is sufficient to lower bound p'.

Lower bound for the probability p'. The probability p' is for an event that's defined as the logical AND of two events, and as usual, equals the product between the probability p'_0 of the first event (the trivial intersection between the subspaces), times the conditional probability p'_1 of the second event (that $\mathcal{A}$ hits a non-zero vector in the dual $T^{\perp}$), conditioned on the first event.

First we lower bound the probability p'_0 by upper bounding the complement probability, that is, we show that the probability for a non-trivial intersection is small. Consider the random process of choosing a basis for a subspace T and note that it is equivalent to choosing a basis for the dual $T^{\perp}$. The process of choosing a basis for the dual has t steps, and in each step we choose a random vector in $S^{\perp}$ that's outside the span we aggregated so far. Given a dual subspace $\overline{T}^{\perp}$ of dimension t, what is the probability for the two subspaces to have only a trivial intersection? It is exactly the sum over $z \in [t]$ (which we think of as the steps for sampling $T^{\perp}$) of the following event: In the t-step process of choosing a basis for $T^{\perp}$, index z was the first to cause the subspaces to have a non-zero intersection. Recall that for each $z \in [t]$, the probability that z was such first index to cause an intersection, equals the probability that the z-th sampled basis vector for $T^{\perp}$ is a vector that's inside the unified span of $\overline{T}^{\perp}$ and the aggregated span of $T^{\perp}$ so far, after $z-1$ samples. This amounts to the probability

$$\sum_{z \in [t]} \frac{|\overline{T}^{\perp}| \cdot 2^{z-1}}{|S^{\perp}|} = \sum_{z \in [t]} \frac{2^t \cdot 2^{z-1}}{2^{k-r}} = 2^{-s} \cdot \sum_{z \in \{0,1,\cdots,t-1\}} 2^z$$

$$= 2^{-s} \cdot \left(2^t - 1\right) < 2^{t-s} \ .$$

Since the above is an upper bound on the probability for a non-trivial intersection between $T^\perp$ and one more single subspace, by union bound, the probability for $T^\perp$ to have a non-trivial intersection with at least one of the t subspaces $\overline{T}_1^\perp, \cdots, \overline{T}_t^\perp$ is upper bounded by $t \cdot 2^{t-s}$. This means that $p'_0 \geq 1 - t \cdot 2^{t-s}$.

The lower bound for the conditional probability p'_1 is now quite easy: Note that since $\Pr[A|B] \geq \Pr[A] - \Pr[\neg B]$, letting A the event that $\mathcal{A}$ outputs a vector in the dual $T^\perp$ and B the event that $T^\perp$ has only a trivial intersection with all other t subspaces, we get $p'_1 \geq \epsilon - t \cdot 2^{t-s}$. By our assumption that $\frac{t \cdot \frac{1}{\epsilon}}{2^{s-t}} \leq o(1)$, we have $p'_1 \geq \frac{\epsilon}{2}$. Overall we got $p' := p'_0 \cdot p'_1 \geq \left(1 - t \cdot 2^{t-s}\right) \cdot \frac{\epsilon}{2} > \frac{\epsilon}{4}$.

Finally, to see why we get an overall high probability for $D \geq t+1$ on a sample from $\mathcal{D}_1$, observe the following. In each bucket there are $\frac{k}{\epsilon}$ attempts, each succeeds with probability at least $\frac{\epsilon}{4}$ and thus the overall success probability in a bucket is $\geq 1 - e^{-\Omega(k)}$. Accordingly, the probability to succeed at least once in each of the $t+1$ buckets (and thus to satisfy $D \geq t+1$) is $\geq 1-(t+1) \cdot e^{-\Omega(k)}$, by considering the complement probability and applying union bound. Overall the probability for $D \geq t+1$ is thus $\geq 1 - e^{-\Omega(k)}$.

Executing $\mathcal{B}$ on the distribution $\mathcal{D}_2$. Consider a different distribution $\mathcal{D}_2$: Sample T once, then sample ℓ i.i.d. obfuscations of the same circuit C_T, denoted $\mathsf{O}_T^{(1)}, \cdots, \mathsf{O}_T^{(\ell)}$. By Corollary 25,

$$\mathcal{D}_1 \approx_{\left(f(\lambda) - \ell \cdot \mathsf{poly}(\lambda),\ \frac{2 \cdot s \cdot \ell}{f(\lambda)}\right)} \mathcal{D}_2 \ .$$

Recall that the running time of $\mathcal{B}$ is $\ell \cdot T_{\mathcal{A}} + \ell \cdot k^3$ and by our Lemma's assumptions, the complexity of $\mathcal{B}$ is $\leq f(\lambda) - \ell \cdot \mathsf{poly}(\lambda)$. Since given a sample oracle from $\mathcal{D}_1$, the algorithm $\mathcal{B}$ outputs $D \geq t+1$ with probability $\geq 1 - e^{-\Omega(k)}$, by the above indistinguishability, whenever we execute $\mathcal{B}$ on a sample from $\mathcal{D}_2$, then with probability at least $\geq 1 - e^{-\Omega(k)} - \frac{2 \cdot s \cdot \ell}{f(\lambda)} \geq 1 - \frac{4 \cdot s \cdot \ell}{f(\lambda)}$ we have $D \geq t+1$. By our assumption in the Lemma that $O\left(\frac{s \cdot \ell}{f(\lambda)}\right) \leq \frac{1}{2}$, with probability at least $\frac{1}{2}$ we have $D \geq t+1$ given a sample from $\mathcal{D}_2$. By an averaging argument, it follows that with probability at least $\frac{1}{2} \cdot \frac{1}{2} = \frac{1}{4}$ over sampling the superspace T, the probability p_T for the event where $D \geq t+1$, is at least $\frac{1}{2} \cdot \frac{1}{2} = \frac{1}{4}$. Let us call this set of superspaces T, "the good set" of samples, which by definition has fraction at least $\frac{1}{4}$. Recall two facts: (1) the dimension of $T^\perp$ is t, (2) The dimension D aggregates vectors inside $S^\perp$. The two facts together imply that in the event $D \geq t+1$, it is necessarily the case that there exists an execution index $i \in [\ell]$ in the reduction $\mathcal{B}$ where $\mathcal{A}$ outputs a vector in $\left(S^\perp \setminus T^\perp\right)$.

For every T inside the good set we thus know that with probability $\frac{1}{4}$, one of the output vectors of $\mathcal{A}$ will be in $\left(S^\perp \setminus T^\perp\right)$. Since these are ℓ i.i.d. executions of $\mathcal{A}$, by union bound, for every T inside the good set, when we prepare an obfuscation O_T of T and execute $\mathcal{A}$, we will get a vector in $\left(S^\perp \setminus T^\perp\right)$ with probability $\frac{1}{4 \cdot \ell}$. We deduce that for a uniformly random T, the probability for $\mathcal{A}(\mathsf{O}_T) \in \left(S^\perp \setminus T^\perp\right)$ is at least the probability for $\mathcal{A}(\mathsf{O}_T) \in \left(S^\perp \setminus T^\perp\right)$ intersecting with the event that T is inside the good set, which in turn is at least

$$\frac{1}{4} \cdot \frac{1}{4 \cdot \ell} = \frac{1}{16 \cdot \ell} := \frac{\epsilon}{16 \cdot k \cdot (t+1)} \ ,$$

which finishes our proof.

□

4 One-Shot Signatures Relative to a Classical Oracle

In this section we present our construction of OSS with respect to a classical oracle and proof of security. We first describe our scheme in 27.

Construction 27. *Let $\lambda \in \mathbb{N}$ the statistical security parameter. Define $s := 16 \cdot \lambda$ and let $n, r, k \in \mathbb{N}$ such that $r := s \cdot (\lambda - 1)$, $n := r + \frac{3}{2} \cdot s$, $k := n$.*

Let $\Pi : \{0,1\}^n \to \{0,1\}^n$ be a random permutation and let $F : \{0,1\}^r \to \{0,1\}^{k \cdot (n-r+1)}$ a random function. Let $H(x)$ denote the first r output bits of $\Pi(x)$, and $J(x)$ denote the remaining $n - r$ bits, which are interpreted as a vector in $\mathbb{Z}_2^{n-r}$. For each $y \in \{0,1\}^r$, let $\mathbf{A}(y) \in \mathbb{Z}_2^{k \times (n-r)}$ be a random matrix with full column-rank, and $\mathbf{b}(y) \in \mathbb{Z}_2^k$ be uniformly random, both are generated by the output randomness of $F(y)$.

Then, we let $\mathcal{P} : \{0,1\}^n \to \left(\{0,1\}^r \times \mathbb{Z}_2^k\right)$, $\mathcal{P}^{-1} : \left(\{0,1\}^r \times \mathbb{Z}_2^k\right) \to \{0,1\}^n$, $\mathcal{D} : \left(\{0,1\}^r \times \mathbb{Z}_2^k\right) \to \{0,1\}$ be the following oracles:

$$\mathcal{P}(x) = \left(y \ , \ \mathbf{A}(y) \cdot J(x) + \mathbf{b}(y) \right) \ \text{ where } y = H(x)$$

$$\mathcal{P}^{-1}(y, \mathbf{u}) = \begin{cases} \Pi^{-1}(y, \mathbf{z}) & \exists\, \mathbf{z} \in \mathbb{Z}_2^{n-r} \text{ such that } \mathbf{A}(y) \cdot \mathbf{z} + \mathbf{b}(y) = \mathbf{u} \\ \perp & \text{else} \end{cases}$$

$$\mathcal{D}(y, \mathbf{v}) = \begin{cases} 1 & \text{if } \mathbf{v}^T \cdot \mathbf{A}(y) = 0^{n-r} \\ 0 & \text{otherwise} \end{cases}$$

We will denote the above distribution of oracles by $\mathcal{O}_{n,r,k}$. We define our hash function as H, which can be easily computed by querying $\mathcal{P}$, considering only the first r bits and discarding the second output.

Note that since $\mathbf{A}(y) \in \mathbb{Z}_2^{k \times (n-r)}$ is full column-rank, $\mathbf{z} \in \mathbb{Z}_2^{n-r}$ is unique if it exists. Thus, $\mathcal{P}^{-1}(\mathcal{P}(x)) = x$ and $\mathcal{P}^{-1}(y, \mathbf{u}) = \perp$ if $(y, \mathbf{u})$ is not in the range of $\mathcal{P}$. Thus, $\mathcal{P}^{-1}$ is the uniquely-defined inverse of the injective function $\mathcal{P}$.

Efficient Implementation. Given the description of Construction 27, it is straightforward to implement it, using pseudorandom functions and pseudorandom permutations, the existence of both of which follow from the existence of one-way functions [Zha21, Zha16]. Specifically, we swap the truly random permutation Π with a pseudorandom permutation PRP, and swap the truly random function F with a pseudorandom function PRF. One can easily verify that under the security of PRP, PRF for quantum queries, the classical efficient construction is computationally indistinguishable from the oracle distribution $\mathcal{O}_{n,r,k}$ described in Construction 27.

Non-collapsing. We prove that our hash function is non-collapsing.

Proposition 28 (Non-collapsing of H)**.** *The hash function H defined in Construction 27 is (always) non-collapsing, as per Definition 10.*

Proof. We explain the non-collapsing property of our scheme by describing the algorithms $(\mathcal{S}_H, \mathcal{D}_H)$. The first algorithm $\mathcal{S}_H$ simply computes a uniform superposition $|+\rangle^{\otimes n}$ over n qubits (where n is the input size for H), and outputs it as $|\psi\rangle$. The second algorithm $\mathcal{D}_H$, given an unknown n-qubit state $|\phi\rangle := \sum_{x \in \mathbb{Z}_2^n} \alpha_x \cdot |x\rangle$, acts as follows.

1. Execute $\mathcal{P}$ in superposition to obtain
$$\sum_{x\in\mathbb{Z}_2^n} \alpha_x \cdot |x\rangle|y_x\rangle|\mathbf{u}_x\rangle \ .$$

2. Execute $\mathcal{P}^{-1}$ in superposition to un-compute the input register holding x, to obtain
$$\sum_{x\in\mathbb{Z}_2^n} \alpha_x \cdot |y_x\rangle|\mathbf{u}_x\rangle \ .$$

3. Execute a k-qubit Quantum Fourier Transform over $\mathbb{Z}_2$ on the rightmost k-qubit register (holding the vectors $\mathbf{u}_x$). This boils down to the execution of parallel Hadamard gates $H^{\otimes k}$, to obtain
$$\sum_{x\in\mathbb{Z}_2^n} \alpha_x \cdot |y_x\rangle \left(H^{\otimes k} \cdot |\mathbf{u}_x\rangle\right) \ .$$

4. Execute $\mathcal{D}(\cdot,\cdot)$ in superposition on the state and measure the output bit register. The output of $\mathcal{D}_H$ is identical to the output of $\mathcal{D}(\cdot,\cdot)$.

To finish the explanation for non-collapsing we will show that the pair $(\mathcal{S}_H, \mathcal{D}_H)$ gives a distinguishing advantage of $\geq 1-2^{-\Omega(\lambda)}$, between the cases of full measurement and partial measurement.

- In the first case, given $|\psi\rangle := |+\rangle^{\otimes n}$, it is measured in the computational basis and the algorithm $\mathcal{D}_H$ gets as input some classical $x \in \mathbb{Z}_2^n$. At the end of Step 2 of $\mathcal{D}_H$ the state is $|y_x\rangle|\mathbf{u}_x\rangle$. The execution of QFT in the next step will generate a uniform superposition over all elements in $\mathbb{Z}_2^k$, at the end of Step 3. The set of elements that are accepted by $\mathcal{D}(y_x,\cdot)$ (for every classical y_x) is a coset of dimension r and thus in particular a set of size 2^r. The probability that the k-qubit uniform superposition will be accepted by $\mathcal{D}(y_x,\cdot)$ is $\frac{2^r}{2^k} = 2^{-(k-r)} \leq 2^{-\Omega(\lambda)}$. It follows that the probability that $\mathcal{D}_H$ outputs 1 in the first case is $\leq 2^{-\Omega(\lambda)}$.

- In the second case, given $|\psi\rangle := |+\rangle^{\otimes n}$ we compute $\mathcal{P}$ in superposition and measure a value y on the side. The algorithm $\mathcal{D}_H$ thus gets as input the state $\sum_{x\in\mathbb{Z}_2^n : H(x)=y} \sqrt{2^{-(n-r)}} \cdot |x\rangle$ for some y. One can easily verify that at the end of Step 2, the state that $\mathcal{D}_H$ holds is
$$|y\rangle \otimes \left(\sum_{\mathbf{u}\in \mathsf{ColSpan}(\mathbf{A}(y))} \sqrt{|\mathsf{ColSpan}\left(\mathbf{A}(y)\right)|^{-1}} \cdot |\mathbf{u}+\mathbf{b}(y)\rangle \right) \ .$$
Next, by standard known properties of QFT applied to a quantum state that is in a uniform superposition over a coset, it follows that at the end of Step 3, the state that $\mathcal{D}_H$ holds is
$$|y\rangle \otimes \left(\sum_{\mathbf{v}\in \mathsf{ColSpan}(\mathbf{A}(y))^\perp} \sqrt{|\mathsf{ColSpan}\left(\mathbf{A}(y)\right)^\perp|^{-1}} \cdot (-1)^{\langle \mathbf{b}(y),\mathbf{v}\rangle} \cdot |\mathbf{v}\rangle \right) \ .$$
It follows that the probability that $\mathcal{D}_H$ outputs 1 in the second case is 1.

□

Security. Most of this section will be devoted to proving security. There are two main steps to our security proof. The first part of the proof will show Theorem 32 (Proved in Sections 4.1 and 4.2), which says that an adversary $\mathcal{A}$ that manages to find a collision in H given access to $(\mathcal{P}, \mathcal{P}^{-1}, \mathcal{D})$ sampled from $\mathcal{O}_{n,r,k}$, can be transformed to an adversary $\mathcal{B}$ that finds a collision in H in the dual-free setting, i.e., having only access to $(\mathcal{P}, \mathcal{P}^{-1})$ and no access to the dual verification oracle $\mathcal{D}$. The second part of security will show Theorems 34 and 35 (both proved in Section 4.3), which together, show how a collision finder for the dual-free case can be turned into a collision finder for plain 2-to-1 random functions. We obtain the following main security theorem.

Theorem 29 (Collision Resistance of H)**.** *Let $\mathcal{O}_{n,r,k}$ the distribution over oracles defined in Construction 27. Let $\mathcal{A}$ an oracle aided q-query (computationally unbounded) quantum algorithm. Then,*

$$\Pr\left[x_0 \neq x_1 \wedge H(x_0) = H(x_1) \ : \ \begin{matrix}(\mathcal{P}, \mathcal{P}^{-1}, \mathcal{D}) & \leftarrow \mathcal{O}_{n,r,k} \\ (x_0, x_1) & \leftarrow \mathcal{A}^{\mathcal{P}, \mathcal{P}^{-1}, \mathcal{D}}\end{matrix}\right] \leq O\left(\frac{\lambda^3 \cdot k^3 \cdot q^3}{2^\lambda}\right) \ .$$

Proof. Assume towards contradiction that there is an oracle aided quantum algorithm $\mathcal{A}$, making q queries, that given a sample oracle $(\mathcal{P}, \mathcal{P}^{-1}, \mathcal{D}) \leftarrow \mathcal{O}_{n,r,k}$ outputs a collision (x_0, x_1) in H with probability ϵ, such that $\epsilon \geq \omega\left(\frac{\lambda^3 \cdot k^3 \cdot q^3}{2^\lambda}\right)$.

Note that by our parameter choices in Construction 27 and by our assumption towards contradiction $\epsilon \geq \omega\left(\frac{\lambda^3 \cdot k^3 \cdot q^3}{2^\lambda}\right)$, one can verify through calculation that (1) for $s - (n - r - s) := s'$ we have $\frac{k^3 \cdot q^3 \cdot \frac{1}{\epsilon^2}}{2^{s'}} \leq o(1)$ and also (2) $\frac{k^9 \cdot q^7 \cdot \frac{1}{\epsilon^4}}{\sqrt{2^{n-r-s}}} \leq o(1)$. This means that the conditions of Theorem 32 are satisfied, and it follows there is a q-query algorithm $\mathcal{B}$ that gets access only to $(\mathcal{P}, \mathcal{P}^{-1})$, sampled from $(\mathcal{P}, \mathcal{P}^{-1}, \mathcal{D}) \leftarrow \mathcal{O}_{r+s,\, r,\, k-(n-r-s)}$ that finds collisions in H with probability $\geq \frac{\epsilon}{2^6 \cdot k^2}$.

By Theorem 34, it follows there is a q-query algorithm $\mathcal{B}'$ that given oracle access to any $Q : \{0,1\}^{r+s} \to \{0,1\}^r$ an $(r+s,\, r,\, s)$-coset partition function (as per Definition 33), finds a collision in Q with probability $\frac{\epsilon}{2^6 \cdot k^2}$.

The above is in particular true for any distribution $Q \leftarrow \mathcal{Q}$ over $(r+s,\, r,\, s)$-coset partition functions. Since our parameter choices in Construction 27 imply $s \mid (r+s)$, we can consider the distribution $\mathcal{Q}$ over $(r+s,\, r,\, s)$-coset partition functions generated by Theorem 35. It follows by Theorem 35 that

$$\frac{\epsilon}{2^6 \cdot k^2} \leq O\left(\frac{s^3 \cdot q^3}{2^{\frac{(r+s)}{s}}}\right) \ ,$$

which in turn implies

$$\epsilon \leq O\left(\frac{k^2 \cdot \lambda^3 \cdot q^3}{2^\lambda}\right) \ ,$$

in contradiction to $\epsilon \geq \omega\left(\frac{\lambda^3 \cdot k^3 \cdot q^3}{2^\lambda}\right)$. □

4.1 Bloating the Dual

Let $\mathcal{O}'_{n,r,k,s}$ denote the following distribution over $\mathcal{P}, \mathcal{P}^{-1}, \mathcal{D}'$. The primal oracles $\mathcal{P}, \mathcal{P}^{-1}$ are defined identically to $\mathcal{O}_{n,r,k}$. However, now, for $s \leq n - r$, we let $\mathbf{A}(y)^{(0)} \in \mathbb{Z}_2^s$ denote the first s columns

of $\mathbf{A}(y) \in \mathbb{Z}_2^{n-r}$ and $\mathbf{A}(y)^{(1)} \in \mathbb{Z}_2^{n-r-s}$ denote the remaining $n-r-s$ columns. Then, define $\mathcal{D}'$ as the oracle:

$$\mathcal{D}'(y, \mathbf{v}) = \begin{cases} 1 & \text{if } \mathbf{v}^T \cdot \mathbf{A}(y)^{(1)} = 0^{n-r-s} \\ 0 & \text{otherwise} \end{cases}$$

Observe that if $\mathbf{v}^T \cdot \mathbf{A}(y) = 0^{n-r}$, then $\mathbf{v}^T \cdot \mathbf{A}(y)^{(1)} = 0^{n-r-s}$. Thus $\mathcal{D}'$ accepts all points that are accepted by $\mathcal{D}$, but also accepts additional points as well, namely those for which $\mathbf{v}^T \cdot \mathbf{A}(y)^{(0)} \neq 0^s$ but $\mathbf{v}^T \cdot \mathbf{A}(y)^{(1)} = 0^{n-r-s}$. We call this action, of moving from $\mathcal{D}$ to a more relaxed oracle $\mathcal{D}'$, "bloating the dual".

Lemma 30. *Suppose there is an oracle aided q-query quantum algorithm $\mathcal{A}$ such that*

$$\Pr\left[(y_0 = y_1) \wedge (x_0 \neq x_1) \ : \ \begin{array}{rl} \left(\mathcal{P}, \mathcal{P}^{-1}, \mathcal{D}\right) & \leftarrow \mathcal{O}_{n,r,k} \\ (x_0, x_1) & \leftarrow \mathcal{A}^{\mathcal{P}, \mathcal{P}^{-1}, \mathcal{D}} \\ (y_b, \mathbf{u}_b) & \leftarrow \mathcal{P}(x_b) \end{array}\right] \geq \epsilon \ .$$

Also, let $s \leq n-r$ such that (1) for $s - (n-r-s) := s'$ we have $\frac{k^3 \cdot q^3 \cdot \frac{1}{\epsilon^2}}{2^{s'}} \leq o(1)$ and (2) $\frac{k^9 \cdot q^7 \cdot \frac{1}{\epsilon^4}}{\sqrt{2^{n-r-s}}} \leq o(1)$. Then,

$$\Pr\left[\begin{array}{l} (y_0 = y_1 := y) \wedge \\ (\mathbf{u}_0 - \mathbf{u}_1) \notin \mathsf{ColSpan}\left(\mathbf{A}(y)^{(1)}\right) \end{array} \ : \ \begin{array}{rl} \left(\mathcal{P}, \mathcal{P}^{-1}, \mathcal{D}'\right) & \leftarrow \mathcal{O}'_{n,r,k,s} \\ (x_0, x_1) & \leftarrow A^{\mathcal{P}, \mathcal{P}^{-1}, \mathcal{D}'} \\ (y_b, \mathbf{u}_b) & \leftarrow \mathcal{P}(x_b) \end{array}\right] \geq \frac{\epsilon}{2^6 \cdot k^2} \ .$$

Note that $(\mathbf{u}_0 - \mathbf{u}_1) \notin \mathsf{ColSpan}\left(\mathbf{A}(y)^{(1)}\right)$ means in particular that $\mathbf{u}_0, \mathbf{u}_1$, and hence x_0, x_1, are distinct (this follows because for every $y \in \mathbb{Z}_2^r$, the mapping between preimages x and vectors $\mathbf{u}_x \in (\mathsf{ColSpan}\left(\mathbf{A}(y)\right) + \mathbf{b}(y))$ is bijective). Thus, the second expression means that $\mathcal{A}$ is finding collisions, but these collisions satisfy an even stronger requirement.

Proof. Assume there is an oracle-aided q-query quantum algorithm $\mathcal{A}$ that given oracle access to $\left(\mathcal{P}, \mathcal{P}^{-1}, \mathcal{D}\right) \leftarrow \mathcal{O}_{n,r,k}$ outputs a pair (x_0, x_1) of n-bit strings. Denote by ϵ the probability that x_0, x_1 are both distinct and collide in $H(\cdot)$ (i.e., their y-values are identical). We next define a sequence of hybrid experiments, outputs and success probabilities for them, and explain why the success probability in each consecutive pair is statistically close.

- Hyb_0: The original execution of $\mathcal{A}$.

The process Hyb_0 is the above execution of $\mathcal{A}$ on input oracles $\left(\mathcal{P}, \mathcal{P}^{-1}, \mathcal{D}\right)$. We define the output of the process as (x_0, x_1) and the process execution is considered as successful if x_0, x_1 are both distinct and collide in $H(\cdot)$. By definition, the success probability of Hyb_0 is ϵ.

- Hyb_1: Simulating the oracles using only a bounded number of cosets $(\mathbf{A}(y), \mathbf{b}(y))$, by using small-range distribution.

Consider the function F which samples for every $y \in \mathbb{Z}_2^r$ the i.i.d. coset description $(\mathbf{A}(y), \mathbf{b}(y))$. These cosets are then used in all three oracles $\mathcal{P}$, $\mathcal{P}^{-1}$ and $\mathcal{D}$. The difference between the current hybrid and the previous hybrid is that we swap F with F' which is sampled as follows: We set

$R := \left(300 \cdot q^3\right) \cdot \frac{2^7 \cdot k^2}{\epsilon}$ and for every $y \in \mathbb{Z}_2^r$ we sample a uniformly random $i_y \leftarrow [R]$, then sample for every $i \in [R]$ a coset $\left(\mathbf{A}_i \in \mathbb{Z}_2^{k \times (n-r)}, \mathbf{b}_i \in \mathbb{Z}_2^k\right)$ as usual. For $y \in \mathbb{Z}_2^r$ we define $F'(y) := \left(\mathbf{A}_{i_y}, \mathbf{b}_{i_y}\right)$.

By Theorem A.6 from [AGQY22], it follows that for every quantum algorithm making at most q queries and tries to distinguish between F and F', the distinguishing advantage is bounded by $\frac{300 \cdot q^3}{R} < \frac{\epsilon}{8}$, which means in particular that the outputs of this hybrid and the previous one has statistical distance bounded by $\frac{\epsilon}{8}$. It follows in particular that the success probability of the current hybrid is $:= \epsilon_1 \geq \epsilon - \frac{\epsilon}{8} = \frac{7 \cdot \epsilon}{8}$.

- Hyb_2: Relaxing dual verification oracle to accept a larger subspace, by information-theoretical subspace hiding.

The difference between the current hybrid and the previous hybrid is that in the current hybrid we make the dual verification oracle $\mathcal{D}$ more relaxed, and accept more strings. Specifically, recall that as part of sampling our oracles $\left(\mathcal{P}, \mathcal{P}^{-1}, \mathcal{D}\right)$ we sample the function F' and in particular we sample R i.i.d. cosets: $\left(\mathbf{A}_i \in \mathbb{Z}_2^{k \times (n-r)}, \mathbf{b}_i \in \mathbb{Z}_2^k\right)_{i \in [R]}$. Recall that in the previous hybrid, given input $\left(y \in \mathbb{Z}_2^r, \mathbf{v} \in \mathbb{Z}_2^k\right)$, the oracle $\mathcal{D}$ accepts iff $\mathbf{v} \in \mathsf{ColSpan}\left(\mathbf{A}_{i_y}\right)^\perp$. The change we make to the current hybrid is the following and applies only to the dual verification oracle $\mathcal{D}$: For each $i \in [R]$, we sample $T_i^\perp$ which is a uniformly random, $(k-n+r+s)$-dimensional superspace of $\mathsf{ColSpan}\left(\mathbf{A}_i\right)^\perp$ (which has dimension $k-n+r$). When we execute $\mathcal{D}$ we check membership in $T_i^\perp$ rather than in the more restrictive, $(k-n+r)$-dimensional $\mathsf{ColSpan}\left(\mathbf{A}\right)^\perp$.

By Lemma 19, due to $k-(k-(n-r))-s = n-r-s$, for every $i \in [R]$, changing $\mathcal{D}$ to check for membership in $T_i^\perp$ instead of $\mathsf{ColSpan}\left(\mathbf{A}_i\right)^\perp$, is $O\left(\frac{q \cdot s}{\sqrt{2^{n-r-s}}}\right)$-indistinguishable, for any q-query algorithm. Since we use the above indistinguishability R times, we get $R \cdot O\left(\frac{q \cdot s}{\sqrt{2^{n-r-s}}}\right)$-indistinguishability. It follows that the success probability of the current hybrid is $:= \epsilon_2 \geq \epsilon_1 - R \cdot O\left(\frac{q \cdot s}{\sqrt{2^{n-r-s}}}\right) \geq \frac{7 \cdot \epsilon}{8} - O\left(\frac{k^2 \cdot q^4 \cdot s \cdot \frac{1}{\epsilon}}{\sqrt{2^{n-r-s}}}\right)$, which in turn (by our Lemma's assumptions) is at least $\frac{3 \cdot \epsilon}{4}$.

- Hyb_3: Asking for the sum of collisions to be outside of T_i, by using dual-subspace anti-concentration.

In the current hybrid we change the success predicate of the experiment. Recall that as part of sampling the oracles in the previous hybrid, we sample R i.i.d. cosets $(\mathbf{A}_i, \mathbf{b}_i)_{i \in [R]}$ which are used in all three oracles $\left(\mathcal{P}, \mathcal{P}^{-1}, \mathcal{D}\right)$. We then sample R i.i.d. $(k-n+r+s)$-dimensional superspaces $\left(T_i^\perp\right)_{i \in [R]}$ of the R corresponding duals $\left(\mathsf{ColSpan}\left(\mathbf{A}_i\right)^\perp\right)_{i \in [R]}$. The change we make to the success predicate is the following: At the end of the execution we get a pair (x_0, x_1) from $\mathcal{A}$. We define the process as successful if $y_0 = y_1 := y$ and also $(\mathbf{u}_0 - \mathbf{u}_1) \notin T_{i_y}$, rather than only asking that $x_0 \neq x_1$.

Let ϵ_3 be the success probability of the current hybrid and note that $\mathcal{A}$ finds collisions with probability ϵ_2 in the previous hybrid Hyb_2 (and since this hybrid is no different, the same goes for the current hybrid). For every value $i \in [R]$ denote by $\epsilon_2^{(i)}$ the probability to find a collision in index i, or formally, that $y_0 = y_1 := y$, $x_0 \neq x_1$ and also $i_y = i$. Observe that in such an event we also have $(\mathbf{u}_0 - \mathbf{u}_1) \in S_{i_y}$. We deduce $\sum_{i \in [R]} \epsilon_2^{(i)} = \epsilon_2$. Let L be a subset of indices $i \in [R]$ such that $\epsilon_2^{(i)} \geq \frac{\epsilon_2}{2 \cdot R}$ and note that $\sum_{i \in L} \epsilon_2^{(i)} \geq \frac{\epsilon_2}{2}$. For every value $i \in [R]$ also denote by $\epsilon_3^{(i)}$ the probability that $y_0 = y_1 := y$, $(\mathbf{u}_0 - \mathbf{u}_1) \notin T_{i_y}$ and also $i_y = i$. We deduce $\sum_{i \in [R]} \epsilon_3^{(i)} = \epsilon_3$.

We would now like to use Lemma 22, so we make sure that we satisfy its requirements. Let any $i \in L$, we know that by definition $\epsilon_2^{(i)} \geq \frac{\epsilon_2}{2 \cdot R}$ and also recall that $\epsilon_2 \geq \frac{3 \cdot \epsilon}{4}$, $R := \left(300 \cdot q^3\right) \cdot \frac{2^7 \cdot k^2}{\epsilon}$ and thus

$$\epsilon_2^{(i)} \geq \frac{\epsilon_2}{2 \cdot R} \geq \frac{3 \cdot \epsilon}{8} \cdot \frac{1}{R} \geq \Omega\left(\frac{\epsilon^2}{q^3 \cdot k^2}\right) \quad .$$

Let $s' := s - (n - r - s)$ and for any $i \in L$ let $\ell_i := \frac{k^2}{\epsilon_2^{(i)}} \leq O\left(\frac{k^4 \cdot q^3}{\epsilon^2}\right)$. Note that by our Lemma 30 statement's assumptions, we have (1) $\frac{(n-r-s) \cdot \frac{1}{\epsilon_2^{(i)}}}{2^{s'}} \leq o(1)$ and (2) $\frac{q \cdot \ell_i^2 \cdot s}{\sqrt{2^{n-r-s}}} \leq o(1)$. Since this satisfies Lemma 22, it follows that for every $i \in L$ we have $\epsilon_3^{(i)} \geq \frac{\epsilon_2^{(i)}}{16 \cdot k^2}$. It follows that

$$\epsilon_3 = \sum_{i \in [R]} \epsilon_3^{(i)} \geq \sum_{i \in L} \epsilon_3^{(i)} \geq \sum_{i \in L} \frac{\epsilon_2^{(i)}}{16 \cdot k^2} \geq \frac{\left(\frac{\epsilon_2}{2}\right)}{16 \cdot k^2} \geq \frac{3 \cdot \epsilon}{2^7 \cdot k^2} \quad .$$

- Hyb_4: For every $i \in [R]$, de-randomizing T_i and defining it as the column span of $\mathbf{A}_i^{(1)} \in \mathbb{Z}_2^{k \times (n-(r+s))}$, the last $n-(r+s)$ columns of the matrix $\mathbf{A}_i$, by using the random permutation Π and random function F.

This hybrid is the same as the previous, with one change: For every $i \in [R]$, after sampling the coset $(\mathbf{A}_i, \mathbf{b}_i)$, we will not continue to randomly sample $T_i^{\perp}$ (which, previously, was a uniformly random $((k-(n-r))+s)$-dimensional superspace of $\mathsf{ColSpan}\left(\mathbf{A}_i\right)^{\perp}$) and simply define $T_i := \mathsf{ColSpan}\left(\mathbf{A}_i^{(1)}\right)$ such that $\mathbf{A}_i^{(1)} \in \mathbb{Z}_2^{k \times (n-r-s)}$ is defined to be the last $n-r-s$ columns of the matrix $\mathbf{A}_i \in \mathbb{Z}_2^{k \times (n-r)}$. We will define an intermediate hybrid $\mathsf{Hyb}_{3.1}$ and then explain why $\mathsf{Hyb}_3 \equiv \mathsf{Hyb}_{3.1} \equiv \mathsf{Hyb}_4$.

- **Using the random permutation Π.** For every $i \in [R]$ consider the superspace $T_i^{\perp}$, which has $(k-n+r)+s$ dimensions. Accordingly, the dual T_i has $k-((k-n+r)+s)$ dimensions, which equals $n-r-s$. For $i \in [R]$, $j \in [n-r-s]$, denote by $\mathbf{t}_{i,j} \in \mathbb{Z}_2^k$ the j-th basis vector for the subspace T_i. Now, for every $i \in [R]$ let $M_i \in \mathbb{Z}_2^{(n-r) \times (n-r)}$ a full-rank matrix that represents the coordinates vectors of the basis vectors of T_i. Formally, M_i is such that the j-th column of M_i, denoted $M_{i,j} \in \mathbb{Z}_2^{n-r}$, satisfies $\mathbf{A}_i \cdot M_{i,j} = \mathbf{t}_{i,j} \in \mathbb{Z}_2^k$.

  In $\mathsf{Hyb}_{3.1}$, we define the permutation Γ over $\{0,1\}^n$ defined as follows: For an input $s \in \{0,1\}^n$, it takes the left r bits denoted $y \in \mathbb{Z}_2^r$, computes $i_y \in [R]$, then applies matrix multiplication by M_{i_y} to the remaining right $n-r$ bits. Observe that since M_i is full rank for all i, then Γ is indeed a permutation. The change we make from Hyb_3 to $\mathsf{Hyb}_{3.1}$ is that in the current hybrid we apply Γ to the *output* of Π inside the execution of a query to $\mathcal{P}$, and apply Γ^{-1} to the *input* of Π^{-1} inside the execution of a query to $\mathcal{P}^{-1}$. Note that for a truly random n-bit permutation Π, concatenating any fixed permutation Γ like this is statistically equivalent to just computing Π and Π^{-1}, thus the outputs (and in particular success probabilities) between Hyb_3 and $\mathsf{Hyb}_{3.1}$ are identical.

- **Using the random function F.** In Hyb_4, we stop applying the permutation Γ to the output of Π (and likewise stop applying Γ^{-1} to the input of Π^{-1}), and also stop sampling $T_i^{\perp}$ and simply define it as $\mathsf{ColSpan}\left(\mathbf{A}_i^{(1)}\right)^{\perp}$. Note that for every choice of Π, the following

two distributions over oracles are statistically equivalent. (1) for every $i \in [R]$ sample $\mathbf{A}_i$ uniformly at random, then sample the superspace $T_i^{\perp}$ uniformly at random, and then set Γ accordingly and concatenate it with Π as we did in the above $\mathsf{Hyb}_{3.1}$. (2) Just sample $\mathbf{A}_i$ uniformly at random for every $i \in [R]$, and then set $T_i := \mathsf{ColSpan}\left(\mathbf{A}_i^{(1)}\right)$. It follows that due to the fact that F is a random function (or more precisely, because for every $i \in [R]$ we choose a uniformly random coset $(\mathbf{A}_i, \mathbf{b}_i)$) then the outputs of $\mathsf{Hyb}_{3.1}$ and Hyb_4 are statistically equivalent and in particular the success probability in both cases is the same.

It follows that the success probability ϵ_4 in Hyb_4 equals the success probability ϵ_3 in Hyb_3.

- Hyb_5: Moving back to using an exponential number of cosets, by using small-range distribution again.

We rewind the process of sampling an R-small range distribution version of F, and use F as a standard random function. By the same argument for the indistinguishability between Hyb_0 and Hyb_1, the output of the current process has statistical distance bounded by $\frac{300 \cdot q^3}{R} = \frac{\epsilon}{2^7 \cdot k^2}$, which means in particular that the outputs of this hybrid and the previous hybrid has statistical distance bounded by $\frac{\epsilon}{2^7 \cdot k^2}$. It follows that the success probability of the current hybrid is

$$:= \epsilon_5 \geq \epsilon_4 - \frac{\epsilon}{2^7 \cdot k^2} \geq \frac{3 \cdot \epsilon}{2^7 \cdot k^2} - \frac{\epsilon}{2^7 \cdot k^2} = \frac{\epsilon}{2^6 \cdot k^2} \ .$$

To conclude, note that the process Hyb_5 is exactly the process where $\mathcal{A}$ executes on input oracle sampled from $\left(\mathcal{P}, \mathcal{P}^{-1}, \mathcal{D}'\right) \leftarrow \mathcal{O}'_{n,r,k,s}$. This finishes our proof. □

4.2 Simulating the Dual

In this section we prove the following lemma.

Lemma 31. *Suppose there is an oracle aided q-query quantum algorithm $\mathcal{A}$ such that*

$$\Pr\left[\begin{array}{l} y_0 = y_1 := y, \\ (\mathbf{u}_0 - \mathbf{u}_1) \notin \mathsf{ColSpan}\left(\mathbf{A}(y)^{(1)}\right) \end{array} : \begin{array}{rl} \left(\mathcal{P}, \mathcal{P}^{-1}, \mathcal{D}'\right) & \leftarrow \mathcal{O}'_{n,r,k,s} \\ (x_0, x_1) & \leftarrow \mathcal{A}^{\mathcal{P}, \mathcal{P}^{-1}, \mathcal{D}'} \\ (y_b, \mathbf{u}_b) & \leftarrow \mathcal{P}(x_b) \end{array}\right] \geq \epsilon \ .$$

Then, there is an oracle aided q-query quantum algorithm $\mathcal{B}$ such that

$$\Pr\left[(\overline{y}_0 = \overline{y}_1 := \overline{y}) \wedge (\overline{x}_0 \neq \overline{x}_1) \ : \begin{array}{rl} \left(\overline{\mathcal{P}}, \overline{\mathcal{P}}^{-1}, \overline{\mathcal{D}}\right) & \leftarrow \mathcal{O}_{r+s,\, r,\, k-(n-r-s)} \\ (\overline{x}_0, \overline{x}_1) & \leftarrow \mathcal{B}^{\overline{\mathcal{P}}, \overline{\mathcal{P}}^{-1}} \\ (\overline{y}_b, \overline{\mathbf{u}}_b) & \leftarrow \overline{\mathcal{P}}(x_b) \end{array}\right] \geq \epsilon \ .$$

Proof. We first describe the actions of the algorithm $\mathcal{B}$ (which will use the code of $\mathcal{A}$ as part of its machinery) and then argue why it breaks collision resistance with the appropriate probability. Given oracle access to $\overline{\mathcal{P}}, \overline{\mathcal{P}}^{-1}$ which comes from $\left(\overline{\mathcal{P}}, \overline{\mathcal{P}}^{-1}, \overline{\mathcal{D}}\right) \leftarrow \mathcal{O}_{r+s,\, r,\, k-(n-r-s)}$, the algorithm $\mathcal{B}$ does the following:

- Sample a random function $F_{\mathbf{C}}$ that outputs some sufficient (polynomial) amount of random bits on an r-bit input, and sample a random n-bit permutation Γ. Define the following oracles.

- $\left(y \in \mathbb{Z}_2^r,\ \mathbf{u} \in \mathbb{Z}_2^k \right) \leftarrow \mathcal{P}\left(x \in \mathbb{Z}_2^n\right)$:
  - $\left(\overline{x} \in \mathbb{Z}_2^{r+s},\ \widetilde{x} \in \mathbb{Z}_2^{n-r-s} \right) \leftarrow \Gamma(x)$.
  - $\left(y \in \mathbb{Z}_2^r,\ \overline{\mathbf{u}} \in \mathbb{Z}_2^{k-(n-r-s)} \right) \leftarrow \overline{\mathcal{P}}(\overline{x})$.
  - $\left(\mathbf{C}(y) \in \mathbb{Z}_2^{k \times k},\ \mathbf{d}(y) \in \mathbb{Z}_2^{n-r-s} \right) \leftarrow F_{\mathbf{C}}(y)$.
  - $\mathbf{u} \leftarrow \mathbf{C}(y) \cdot \begin{pmatrix} \overline{\mathbf{u}} \\ \widetilde{x} + \mathbf{d}(y) \end{pmatrix}$.
- $\left(x \in \mathbb{Z}_2^n \right) \leftarrow \mathcal{P}^{-1}\left(y \in \mathbb{Z}_2^r,\ \mathbf{u} \in \mathbb{Z}_2^k \right)$:
  - $\left(\mathbf{C}(y) \in \mathbb{Z}_2^{k \times k},\ \mathbf{d}(y) \in \mathbb{Z}_2^{n-r-s} \right) \leftarrow F_{\mathbf{C}}(y)$.
  - $\begin{pmatrix} \overline{\mathbf{u}} \\ \widetilde{x} \end{pmatrix} \leftarrow \mathbf{C}(y)^{-1} \cdot \mathbf{u} - \begin{pmatrix} 0^{k-(n-r-s)} \\ \mathbf{d}(y) \end{pmatrix}$.
  - $\left(\overline{x} \in \mathbb{Z}_2^{r+s} \right) \leftarrow \overline{\mathcal{P}}^{-1}(y, \overline{\mathbf{u}})$.
  - $x \leftarrow \Gamma^{-1}(\overline{x}, \widetilde{x})$.
- $\mathcal{D}'\left(y \in \mathbb{Z}_2^r,\ \mathbf{v} \in \mathbb{Z}_2^k \right) \in \{0,1\}$:
  - $\left(\mathbf{C}(y) \in \mathbb{Z}_2^{k \times k},\ \mathbf{d}(y) \in \mathbb{Z}_2^{n-r-s} \right) \leftarrow F_{\mathbf{C}}(y)$.
  - $\mathbf{A}^{(1)}(y) :=$ last $n-r-s$ columns of $\mathbf{C}(y)$.
  - Output 1 iff $\mathbf{v}^T \cdot \mathbf{A}^{(1)}(y) = \mathbf{0}^{n-r-s}$.

The remainder of the reduction is simple: $\mathcal{B}$ executes $(x_0, x_1) \leftarrow \mathcal{A}^{\mathcal{P},\mathcal{P}^{-1},\mathcal{D}'}$ and then $(\overline{x}_b, \widetilde{x}_b) \leftarrow \Gamma(x_b)$ and outputs $(\overline{x}_0, \overline{x}_1)$. Assume that the output of $\mathcal{A}$ satisfies $y_0 = y_1 := y$ and also $(\mathbf{u}_0 - \mathbf{u}_1) \notin \mathsf{ColSpan}\left(\mathbf{A}(y)^{(1)}\right)$, and recall that $\mathbf{A}(y)^{(1)} \in \mathbb{Z}_2^{k \times (n-r-s)}$ are the last $n-r-s$ columns of the matrix $\mathbf{A}(y) \in \mathbb{Z}_2^{k \times (n-r)}$, which is generated by the reduction. We explain why it is necessarily the case that $\overline{x}_0 \neq \overline{x}_1$.

First note that due to how we defined the reduction, $\mathbf{A}(y) := \mathbf{C}(y) \cdot \begin{pmatrix} \overline{\mathbf{A}}(y) & \\ & \mathbf{I}_{n-r-s} \end{pmatrix}$, where $\overline{\mathbf{A}}(y) \in \mathbb{Z}_2^{(k-(n-r-s)) \times s}$ is the matrix arising from the oracles $\overline{\mathcal{P}}, \overline{\mathcal{P}}^{-1}$ and $\mathbf{I}_{n-r-s} \in \mathbb{Z}_2^{(n-r-s) \times (n-r-s)}$ is the identity matrix of dimension $n-r-s$. Also note that because $\mathbf{C}(y)$, $\overline{\mathbf{A}}(y)$ are full rank then $\mathbf{A}(y)$ is full rank. Now, since $(\mathbf{u}_0 - \mathbf{u}_1) \notin \mathsf{ColSpan}\left(\mathbf{A}(y)^{(1)}\right)$ and since $\mathbf{A}(y)^{(1)}$ are the last $n-r-s$ columns of $\mathbf{A}(y)$, it follows that if we consider the coordinates vector $\mathbf{x} \in \mathbb{Z}_2^{n-r}$ of $(\mathbf{u}_0 - \mathbf{u}_1)$ with respect to $\mathbf{A}(y)$, the first s elements are not 0^s. By linearity of matrix multiplication it follows that if we look at each of the two coordinates vectors $\mathbf{x}_0, \mathbf{x}_1$ (each has $n-r$ bits) for $\mathbf{u}_0, \mathbf{u}_1$, respectively, somewhere in the first s bits, they differ. Now, recall how we obtain the first s bits of $\mathbf{x}_b$ – this is exactly by applying $\overline{\Pi}$ (the permutation on $\{0,1\}^{r+s}$ arising from the oracles $\overline{\mathcal{P}}, \overline{\mathcal{P}}^{-1}$) to $\overline{x}_b$ and taking the last s bits of the output. Since these bits differ in the output of the permutation, then the preimages have to differ, i.e., $\overline{x}_0 \neq \overline{x}_1$.

Define $\epsilon_{\mathcal{B}}$ as the probability that the output of $\mathcal{A}$ indeed satisfies $y_0 = y_1 := y$ and also $(\mathbf{u}_0 - \mathbf{u}_1) \notin \mathsf{ColSpan}\left(\mathbf{A}^{(1)}(y)\right)$, and it remains to give a lower bound for the probability $\epsilon_{\mathcal{B}}$. We do this by a sequence of hybrids, eventually showing that the oracle which $\mathcal{B}$ simulates to $\mathcal{A}$ is

indistinguishable from an oracle sampled from $\mathcal{O}'_{n,r,k,s}$. More precisely, each hybrid describes a process, it has an output, and a success predicate on the output.

- Hyb_0: The above distribution $\left(\mathcal{P}, \mathcal{P}^{-1}, \mathcal{D}'\right) \leftarrow \mathcal{B}^{\overline{\mathcal{P}}, \overline{\mathcal{P}}^{-1}}$, simulated to the algorithm $\mathcal{A}$.

The first hybrid is where $\mathcal{B}$ executes $\mathcal{A}$ by the simulation described above. The output of the process is the output (x_0, x_1) of $\mathcal{A}$. The process execution is considered as successful if $y_0 = y_1 := y$ and $(\mathbf{u}_0 - \mathbf{u}_1) \notin \mathsf{ColSpan}\left(\mathbf{A}(y)^{(1)}\right)$.

- Hyb_1: Not applying the inner permutation $\overline{\Pi}$ (which comes from the oracles $\overline{\mathcal{P}}$, $\overline{\mathcal{P}}^{-1}$), by using the random permutation Γ.

Let $\overline{\Pi}$ the permutation on $\{0,1\}^{r+s}$ that's inside $\overline{\mathcal{P}}$. In the previous hybrid we apply the n-bit permutation Γ to the input $x \in \mathbb{Z}_2^n$ and then proceed to apply the inner permutation $\overline{\Pi}$ to the first (i.e. leftmost) $r+s$ output bits of the first permutation Γ (we also apply Γ^{-1} to the output of the inverse of the inner permutation, in the inverse oracle $\mathcal{P}^{-1}$). The change we make to the current hybrid is that we simply apply only Γ and discard the inner permutation and its inverse. Since a random permutation concatenated with any permutation distributes identically to a random permutation, the current hybrid is statistically equivalent to the previous and in particular the output of this process distributes identically to the output of the previous, and so does the success probability.

- Hyb_2: For every $y \in \mathbb{Z}_2^r$, taking $\mathbf{A}(y)$ to be the direct output of F, by using the randomness of the random function.

In order to describe the change between the current and previous hybrid we first recall the structure of the oracles from the previous hybrid: Observe that in the previous hybrid, for every $y \in \mathbb{Z}_2^r$ we defined $\mathbf{A}(y) := \mathbf{C}(y) \cdot \begin{pmatrix} \overline{\mathbf{A}}(y) & \\ & \mathbf{I}_{n-r-s} \end{pmatrix}$, where $\mathbf{C}(y) \in \mathbb{Z}_2^{k \times k}$ is the output of $F_{\mathbf{C}}(y)$ and $\overline{\mathbf{A}}(y) \in \mathbb{Z}_2^{(k-(n-r-s)) \times s}$ is the output of the inner random function $\overline{F}$ (which comes from the inside of the oracles $\left(\overline{\mathcal{P}}, \overline{\mathcal{P}}^{-1}\right)$). In the current hybrid we are going to ignore the inner random function $\overline{F}$, its generated matrix $\overline{\mathbf{A}}(y)$ and also the pair $\mathbf{C}(y)$, $\mathbf{d}(y)$, sample a fresh random function $F_{\mathbf{A}}$ at the beginning of the process, and on query y generate $(\mathbf{A}(y), \mathbf{b}(y)) \leftarrow F_{\mathbf{A}}(y)$, for $\mathbf{A}(y) \in \mathbb{Z}_2^{k \times (n-r)}$, $\mathbf{b}(y) \in \mathbb{Z}_2^k$.

To see why the two distributions are indistinguishable, note that the following two ways to sample $\mathbf{A}(y)$, are statistically equivalent: (1) For every $y \in \mathbb{Z}_2^r$, the matrix $\mathbf{A}(y)$ is generated by sampling a random full-rank matrix $\mathbf{C}(y) \in \mathbb{Z}_2^{k \times k}$ and letting $\mathbf{A}(y)$ be $\mathbf{C}(y) \cdot \begin{pmatrix} \overline{\mathbf{A}}(y) & \\ & \mathbf{I}_{n-r-s} \end{pmatrix}$. (2) For every $y \in \mathbb{Z}_2^r$ just sample a full-rank matrix $\mathbf{A}(y) \in \mathbb{Z}_2^{k \times (n-r)}$. Since we are using random functions and in the previous hybrid we are sampling $\mathbf{A}(y)$ according to (1) and in the current hybrid we are sampling $\mathbf{A}(y)$ according to (2), the outputs of the two hybrids distribute identically.

Finalizing the reduction. Observe that the distribution generated in the above Hyb_2 is exactly an oracle sampled from $\mathcal{O}'_{n,r,k,s}$. From the lemma's assumptions, the success probability for Hyb_2 is thus ϵ. Since we also showed that the hybrids have identical success probabilities, it follows that $\epsilon_{\mathcal{B}} = \epsilon$, which finishes our proof. □

We conclude this section by stating the following Theorem, which is obtained as a direct corollary from Lemmas 30 and 31.

Theorem 32. *Suppose there is an oracle aided q-query quantum algorithm $\mathcal{A}$ such that*

$$\Pr\left[x_0 \neq x_1 \wedge H(x_0) = H(x_1) \ : \ \begin{matrix}(\mathcal{P}, \mathcal{P}^{-1}, \mathcal{D}) & \leftarrow \mathcal{O}_{n,r,k} \\ (x_0, x_1) & \leftarrow \mathcal{A}^{\mathcal{P},\mathcal{P}^{-1},\mathcal{D}}\end{matrix}\right] \geq \epsilon \ .$$

Also, let $s \leq n - r$ such that (1) for $s - (n - r - s) := s'$ we have $\frac{k^3 \cdot q^3 \cdot \frac{1}{\epsilon^2}}{2^{s'}} \leq o(1)$ and (2) $\frac{k^9 \cdot q^7 \cdot \frac{1}{\epsilon^4}}{\sqrt{2^{n-r-s}}} \leq o(1)$. Then, there is an oracle aided q-query quantum algorithm $\mathcal{B}$ such that

$$\Pr\left[x_0 \neq x_1 \wedge H(x_0) = H(x_1) \ : \ \begin{matrix}(\mathcal{P}, \mathcal{P}^{-1}, \mathcal{D}) \leftarrow \mathcal{O}_{r+s,\, r,\, k-(n-r-s)} \\ (x_0, x_1) \leftarrow \mathcal{B}^{\mathcal{P},\mathcal{P}^{-1}}\end{matrix}\right] \geq \frac{\epsilon}{2^6 \cdot k^2} \ .$$

4.3 Hardness of the Dual-free Case from 2-to-1 Collision-Resistance

We start with defining coset partition functions, which are an object we will use in order to show that collision finding in the dual-free case is at least as hard as finding collisions in 2-to-1 random functions.

Definition 33 (Coset Partition Functions)**.** *For $n, \ell \in \mathbb{N}$ such that $\ell \leq n$ we say a function $Q : \{0,1\}^n \to \{0,1\}^m$ is a (n, m, ℓ)-*coset partition function *if, for each y in the image of Q, the pre-image set $Q^{-1}(y)$ has size 2^ℓ and is a coset of a linear space of dimension ℓ. We allow different pre-image sets to be cosets of different linear spaces.*

From Dual-free to Coset Partition Functions. We next show that finding collisions in the dual-free case is no easier than finding collisions in random coset partition functions.

Theorem 34. *Let $k \geq n \geq r$. Suppose there is an oracle aided q-query quantum algorithm $\mathcal{A}$ such that*

$$\Pr\left[(y_0 = y_1 := y) \wedge (x_0 \neq x_1) \ : \ \begin{matrix}(\mathcal{P}, \mathcal{P}^{-1}, \mathcal{D}) & \leftarrow \mathcal{O}_{n,r,k} \\ (x_0, x_1) & \leftarrow \mathcal{A}^{\mathcal{P},\mathcal{P}^{-1}} \\ (y_b, \mathbf{u}_b) & \leftarrow \mathcal{P}(x_b)\end{matrix}\right] \geq \epsilon \ .$$

Then there is an oracle aided q-query quantum algorithm $\mathcal{B}$ that given any $(n, r, n-r)$-coset partition function Q, satisfies

$$\Pr\left[(Q(w_0) = Q(w_1)) \wedge (w_0 \neq w_1) \ : \ (w_0, w_1) \leftarrow \mathcal{B}^Q\right] \geq \epsilon \ .$$

Proof. $\mathcal{B}$ works as follows. Given oracle access to some $(n, r, n-r)$-coset partition function Q, it chooses a random permutation $\Gamma : \{0,1\}^n \to \{0,1\}^n$, and for each y, it chooses a random full-column-rank matrix $\mathbf{C}_y \in \mathbb{Z}_2^{k \times n}$ (which is possible since we assume $k \geq n$) and random vector $\mathbf{d}_y \in \mathbb{Z}_2^k$. It then runs $\mathcal{A}$, simulating the oracles $\mathcal{P}, \mathcal{P}^{-1}$ as follows:

The oracle $\mathcal{P}(x)$:

- $y \leftarrow Q(\Gamma(x))$.
- Output $(y, \mathbf{C}_y \cdot \Gamma(x) + \mathbf{d}_y)$.

We now claim that $\mathcal{P}$ is correctly distributed. To do so, we will first define an augmented function $Q' : \{0,1\}^n \to \{0,1\}^n$. On input $\mathbf{z}$, the n-bit output of $Q'(\mathbf{z})$ consists of two parts. The first r bits are set to $y = Q(\mathbf{z})$. The preimage set $Q^{-1}(y)$ is then a coset, which can be described as the set $\{\overline{\mathbf{A}}_y \cdot \mathbf{r} + \overline{\mathbf{b}}_y\}$ as $\mathbf{r}$ ranges over $\mathbb{Z}_2^{n-r}$ (where $\overline{\mathbf{A}}_y$, $\overline{\mathbf{b}}_y$ are both unknown to the reduction algorithm $\mathcal{B}$). Here, $\overline{\mathbf{A}}_y \in \mathbb{Z}_2^{n \times (n-r)}$ has full column-rank and $\overline{\mathbf{b}} \in \mathbb{Z}_2^n$. Define the function $\overline{J}(\mathbf{z})$ that outputs the unique vector in $\mathbb{Z}_2^{n-r}$ such that $\mathbf{z} = \overline{\mathbf{A}}_y \cdot \overline{J}(\mathbf{z}) + \overline{\mathbf{b}}_y$. Then define $Q'(\mathbf{z}) = \left(Q(\mathbf{z}), \overline{J}(\mathbf{z})\right)$. Note that Q' is not efficiently computable without knowing $\overline{\mathbf{A}}_y$,$\overline{\mathbf{b}}_y$, but here we will not need it to be. Intuitively, the reason that we do not need $\overline{J}(\cdot)$ to be efficiently computable is because whenever we simulate the oracles $\left(\mathcal{P}, \mathcal{P}^{-1}, \mathcal{D}\right)$, the value $J(\cdot)$ is never output, it is only the connector between preimages x and vectors $\mathbf{u}_x$ in the coset $\mathsf{ColSpan}(\mathbf{A}(y)) + \mathbf{b}(y)$. Notice that Q' is a function from $\mathbb{Z}_2^n$ to $\mathbb{Z}_2^n$, and it is moreover a permutation with $(Q')^{-1}(y, \mathbf{r}) = \overline{\mathbf{A}}_y \cdot \mathbf{r} + \overline{\mathbf{b}}_y$.

Observe that $\mathcal{B}$'s simulation of $\mathcal{P}$ is implicitly setting the following parameters

$$\Pi(x) = Q'(\Gamma(x)) \,, \qquad H(x) = Q(\Gamma(x)) \,, \qquad J(x) = \overline{J}(\Gamma(x)) \,,$$
$$\mathbf{A}_y = \mathbf{C}_y \cdot \overline{\mathbf{A}}_y \,, \qquad \mathbf{b}_y = \mathbf{C}_y \cdot \overline{\mathbf{b}}_y + \mathbf{d}_y \,.$$

Thus, we must check that these quantities have the correct distribution. Indeed, for every Q, which is in turn defines $\left(\overline{\mathbf{A}}_y, \overline{\mathbf{b}}_y\right)_{y \in \{0,1\}^r}$, the function $Q'(x)$ is a permutation: Given $y \in \{0,1\}^r$, $\mathbf{r} \in \mathbb{Z}_2^{n-r}$, one can recover $\mathbf{z} \in \{0,1\}^n$ as $\mathbf{z} = \overline{\mathbf{A}}_y \cdot \mathbf{r} + \overline{\mathbf{b}}_y$. Hence Π is a permutation since it is the composition of two permutations. Moreover, since one of the two permutations (Γ) is uniformly random, so is Π.

Now we look at the distribution of $\mathbf{A}_y, \mathbf{b}_y$. Recall that $\overline{\mathbf{A}}_y \in \mathbb{Z}_2^{n \times (n-r)}$ is a full-column-rank matrix, and $\mathbf{C}_y \in \mathbb{Z}_2^{k \times n}$ is a *random* full-column-rank matrix. Thus, $\mathbf{A}_y = \mathbf{C}_y \cdot \overline{\mathbf{A}}_y \in \mathbb{Z}_2^{k \times (n-r)}$ is also a random full-column-rank matrix.

Then we have that $\mathbf{b}_y = \mathbf{C}_y \cdot \overline{\mathbf{b}}_y + \mathbf{d}_y$ where $\mathbf{d}_y$ is random, meaning $\mathbf{b}_y$ is random. Thus, $\mathcal{P}$ has an identical distribution to that arising from $\mathcal{O}_{n,r,k}$.

The oracle $\mathcal{P}^{-1}(y, \mathbf{u})$:

- $x \leftarrow \begin{cases} \Gamma^{-1}(\mathbf{w}) & \exists \mathbf{w} \in \mathbb{Z}_2^n \text{ such that } \mathbf{C}_y \cdot \mathbf{w} + \mathbf{d}_y = \mathbf{u} \\ \perp & \text{if no such } \mathbf{w} \text{ exists} \end{cases}$
- Output $\begin{cases} x & \text{if } x \neq \perp \text{ and } Q(\Gamma(x)) = y \\ \perp & \text{if } x = \perp \text{ or } Q(\Gamma(x)) \neq y \end{cases}$

Observe that $\mathcal{P}^{-1}(\mathcal{P}(x)) = x$, and for all pairs $\left(y \in \{0,1\}^r, \mathbf{u} \in \mathbb{Z}_2^k\right)$ that are not in the image of $\mathcal{P}$, we have $\mathcal{P}^{-1}(y, \mathbf{u}) = \perp$. Thus, $\mathcal{P}^{-1}$ is the uniquely-defined inverse of $\mathcal{P}$. Thus, since the distribution of $\mathcal{P}$ simulated by $\mathcal{B}$ exactly matches the distribution arising from $\mathcal{O}_{n,r,k}$, the same is true of the pairs $\left(\mathcal{P}, \mathcal{P}^{-1}\right)$.

Finishing touches. Thus we saw that for every input $(n, r, n-r)$-coset partition function Q, the algorithm $\mathcal{B}$ perfectly simulates the view of $\mathcal{A}$, which consists of the pair of oracles $\mathcal{P}, \mathcal{P}^{-1}$ that distribute according to $\mathcal{O}_{n,r,k}$. Hence, with probability ϵ, the algorithm $\mathcal{A}$ will produce a collision $x_0 \neq x_1$ such that $H(x_0) = H(x_1)$. It remains to explain what $\mathcal{B}$ does in order to obtain a collision in Q, for every collision in the simulated H. Given (x_0, x_1), the reduction $\mathcal{B}$ will then compute and output $(w_0 = \Gamma(x_0), w_1 = \Gamma(x_1))$. Observe that if $x_0 \neq x_1$, then $w_0 \neq w_1$ since Γ is a permutation. Moreover, if $H(x_0) = H(x_1)$, then

$$Q(w_0) = Q\left(\Gamma(x_0)\right) = H(x_0) = H(x_1) = Q\left(\Gamma(x_1)\right) = Q(w_1) \ .$$

Hence, with probability at least ϵ, $\mathcal{B}$ will output a collision for Q. Notice that for each query that $\mathcal{A}$ makes to $\mathcal{P}(\cdot)$, the reduction $\mathcal{B}$ needs to make exactly one query to Q and the same goes for the inverse $\mathcal{P}^{-1}(\cdot)$. Thus $\mathcal{B}$ makes exactly q queries to Q. This completes the proof. □

Collision-resistant Coset Partition Functions. We now show how to construct a collision resistant coset partition functions relative to an oracle. Our main theorem for this subsection is the following.

Theorem 35. *For any $n, \ell \in \mathbb{N}$ such that $\ell \mid n$, there exists a distribution over $(n, n-\ell, \ell)$-coset partition functions H, such that any algorithm making q queries to H can only find collisions in H with probability at most $O\left(\frac{\ell^3 \cdot q^3}{2^{\frac{n}{\ell}}}\right)$.*

Proof. We will start with a much weaker goal of constructing distributions of collision-resistant 2-to-1 functions, that are shrinking by a single bit.

Lemma 36. *A random 2-to-1 function $H : \{0,1\}^n \to \{0,1\}^{n-1}$ is collision resistant given quantum queries to H. In particular, any quantum algorithm making q queries has a $O(q^3/2^n)$ probability of producing a collision.*

Proof. The original work to lower-bound the quantum query complexity of collision resistance was [AS04]. That work proves that random 2-to-1 functions are collision-resistant, provided that the range is at least as large as the domain. They need the large range since their proof works via showing that the function is indistinguishable from a 1-to-1 function. But of course, a 1-to-1 function must have a range at least as large as domain. This is not quite good enough for us, as we insist on our H "losing" one bit.

Instead, we first point out that H can be extended into a permutation as follows. First let $J : \{0,1\}^n \to \{0,1\}$ be an arbitrary function which for any collision (x_0, x_1) of H, assigned random but distinct values to $J(x_0)$ and $J(x_1)$. Since H is 2-to-1, such a J only needs a 1-bit range. Then $\Pi(x) = (H(x), J(x))$ is a permutation, and if H is a random 2-to-1 function, then Π is a random permutation.

We will argue that H is collision-resistant *even given queries to* Π (but not Π^{-1}). Note that this is potentially stronger than giving access to H, as an algorithm can always ignore J. Suppose there is an oracle aided algorithm $\mathcal{A}$ which given oracle access to Π, finds a collision in H (that is, a collision in the first $n-1$ output bits of Π) with probability ϵ.

To argue for the collision resistance given Π, we recall that for every q-query quantum algorithm having access to Π (but not its inverse), Π being a random permutation is $O\left(\frac{q^3}{2^n}\right)$-indistinguishable from Π being a random function [Zha15]. So now let H be the first $n-1$ bits of a random function

$\Pi : \{0,1\}^n \to \{0,1\}^n$, and J be the last bit. Then we have that $\mathcal{A}^{\Pi}$ still finds a collision in this H with probability at least $\epsilon - O\left(\frac{q^3}{2^n}\right)$.

But now we observe that H and J are simply just independent random functions, and J can be simulated without making any queries to H (this can even be made efficient, but that is irrelevant since we only care about query counts). Thus, we obtain an algorithm $\mathcal{B}^H$ which finds a collision in a random function $H : \{0,1\}^n \to \{0,1\}^{n-1}$ with probability at least $\epsilon - O\left(\frac{q^3}{2^n}\right)$. But we know that random functions are collision resistant, regardless of the relationship between domain and range. In particular, the probability of finding a collision in such random H is at most $O\left(\frac{q^3}{2^{n-1}}\right) = O\left(\frac{q^3}{2^n}\right)$ by [Zha15]. Thus, we can bound $\epsilon \leq O\left(\frac{q^3}{2^n}\right)$. □

Extending to coset partition functions. Observe that a 2-to-1 function is trivially a coset partition function. Indeed, since the pre-image set always is a pair $\{x_0, x_1\}$, which is a coset of the 1-dimensional linear space $\{0, x_0 \oplus x_1\} \subseteq \mathbb{Z}_2^n$ over the field $\mathbb{Z}_2$. To conclude what we saw so far, a random 2-to-1 function $H : \{0,1\}^n \to \{0,1\}^{n-1}$ (that is shrinking by one bit) is collision resistant by Lemma 36, and is also a $(n, n-1, 1)$ coset partition function, which proves Theorem 35 for the case $\ell = 1$.

We extend to (an almost) general ℓ as follows. We can let $H^\ell : \{0,1\}^{n\cdot\ell} \to \{0,1\}^{(n-1)\cdot\ell}$ be the function such that

$$H^\ell(x_1, \cdots, x_\ell) := (H(x_1), \cdots, H(x_\ell)) \quad .$$

It is straightforward that any collision for H^ℓ immediately gives a collision for H. Moreover, we can simulate any query to H^ℓ given ℓ queries to H. Thus, any algorithm making q queries to H^ℓ can only find collisions in H^ℓ with probability at most $O\left(\frac{\ell^3 \cdot q^3}{2^n}\right)$.

Lastly, parallel repetition preserves coset partitions, since the pre-image sets of H^ℓ are just the direct sums of ℓ of the pre-image sets of H. Thus, we obtain the theorem by replacing $n \cdot \ell$ with n. □

5 Permutable PRPs and Applications

In this section, we develop a new concept of pseudo-random permutation that will be useful for our obfuscation-based construction. Quite roughly, our notion of a permutation will allow us to compose the permutation with another (fixed) permutation, while hiding that the fixed permutation had been applied.

Definition 37. *Let $G = \{G_N\}_{N\in\mathbb{N}}$ be a collection where each G_N is a set of permutations over $[N]$. An output-permutable PRP (OP-PRP) for G is a tuple of algorithms $\left(\Pi, \Pi^{-1}, \mathsf{Permute}, \mathsf{Eval}, \mathsf{Eval}^{-1}\right)$ with the following properties.*

- **Efficient Permutations:** *For any key $k \in \{0,1\}^\lambda$ and any desired "block size" N, $\Pi(k,\cdot)$ is an efficiently computable permutation on $[N]$ with $\Pi^{-1}(k,\cdot)$ being its efficiently computable inverse.*
- **Output Permuting:** $\mathsf{Permute}(k, \Gamma, c)$ *is a deterministic polynomial-time procedure which takes as input a key $k \in \{0,1\}^\lambda$, the circuit description of a permutation Γ in G_N, and a bit c. It outputs a permuted key $k^{\Gamma,c}$.*

- **Output Permuted Correctness:** *For all* $\lambda \in \mathbb{N}$, $k \in \{0,1\}^\lambda$, $c \in \{0,1\}$, $\Gamma \in G_N$ *and* $x, z \in [N]$,

$$\mathsf{Eval}\left(k^{\Gamma,c}, x\right) = \begin{cases} \Pi(k,x) & \text{if } c = 0 \\ \Gamma\left(\Pi(k,x)\right) & \text{if } c = 1 \end{cases}$$

$$\mathsf{Eval}^{-1}\left(k^{\Gamma,c}, z\right) = \begin{cases} \Pi^{-1}(k,z) & \text{if } c = 0 \\ \Pi^{-1}(k, \Gamma^{-1}(z))) & \text{if } c = 1 \end{cases}$$

We call $k^{\Gamma,c}$ *a permuted key.*

- **Security:** *For any interactive quantum polynomial-time adversary* $\mathcal{A}$*, there exists a negligible function* $\epsilon(\lambda)$ *such that the following experiment with* $\mathcal{A}$ *outputs 1 with probability at most* $\frac{1}{2} + \epsilon(\lambda)$*:*
  - $\mathcal{A}\left(1^\lambda\right)$ *produces a block size* N *(in binary) and the description of a permutation* $\Gamma \in G_N$*.*
  - *The experiment chooses a random* $k \leftarrow \{0,1\}^\lambda$ *and a random bit* $c \in \{0,1\}$*. It gives* $k^{\Gamma,c} \leftarrow \mathsf{Permute}(k, \Gamma, c)$ *to* $\mathcal{A}$*.*
  - $\mathcal{A}$ *produces a guess* c' *for* c*. The experiment outputs 1 if* $c' = c$*.*

On the relations between parameters. Observe that $\lambda, N \in \mathbb{N}$ can be arbitrary, which means in particular that we can fix the block size N and vary the security parameter λ, or alternatively fix λ and vary N. We will often overload notation, and use the same symbol for both Π and Eval (and corresponding symbols for Π^{-1} and Eval^{-1}), when clear from context whether we are using a permuted or normal key. In this case, an OP-PRP would be a triplet $\left(\Pi, \Pi^{-1}, \mathsf{Permute}\right)$.

Sub-exponential Security. For arbitrary functions $f_0, f_1 : \mathbb{N} \to \mathbb{N}$, we say that an OP-PRP is $\left(f_0, \frac{1}{f_1}\right)$-secure if in the above the security part of the definition, we ask that the indistinguishability holds for every adversary of size $\leq f_0(\lambda)$ and we swap $\epsilon(\lambda)$ with $\frac{1}{f_1(\lambda)}$. Concretely, a sub-exponentially secure OP-PRP scheme would be one such that there exists a positive real constant $c > 0$ such that the scheme is $\left(2^{\lambda^c}, \frac{1}{2^{\lambda^c}}\right)$-secure.

From output-permutable to arbitrarily permutable. We say that a PRP is an *input permutable* (IP-) PRP if we apply Γ to the inputs rather than the outputs. Note that by exchanging the roles of Π and Π^{-1} (and likewise Eval and Eval^{-1}), we can turn any OP-PRP into an IP-PRP and vice versa. Also, for a PRP of the form $\Pi_{\mathsf{out}}\left(k_{\mathsf{out}}, \Pi_{\mathsf{in}}\left(k_{\mathsf{in}}, \cdot\right)\right)$, that's the composition of an IP-PRP Π_{in} and an OP-PRP Π_{out}, both properties are simultaneously satisfied. We simply call such a PRP a permutable PRP (P-PRP).

Decomposable Permutations. In this work the class G of permutations that we will be interested in, is the class of *decomposable permutations*, defined next.

Definition 38 (Decomposable Permutations)**.** *Let* $N \in \mathbb{N}$*, let* Γ *a permutation on* $[N]$ *and let* $T, s : \mathbb{N} \to \mathbb{N}$*. We say that* Γ *is* $(T(N), s(N))$*-decomposable if there exists a sequence of permutations* $\Gamma_0, \Gamma_1, \cdots, \Gamma_{T(N)}$ *such that:*

- Γ_0 *is the identity.*
- $\Gamma_{T(N)} = \Gamma$.
- *Each* Γ_i, Γ_i^{-1} *has circuit size at most* $s(N)$.
- *For each* i*, either* $\Gamma_i = \Gamma_{i-1}$ *or there exists a* $z_i \in [N]$ *such that* $\Gamma_i = \Gamma_{i-1} \circ (z_i \;\; z_i + 1)$*, where* $(z_i \;\; z_i + 1)$ *is the neighbor-swap permutation for* z_i*, which swaps between* z_i *and* $(z_i + 1 \bmod N)$*, and acts as the identity on all other elements in* $[N]$.

In the uniform setting, we will additionally ask that there is a uniform polynomial-time (quantum) algorithm which given the description of Γ *and* i *constructs both* z_i *and the circuits for* Γ_i, Γ_i^{-1}.

Examples of Decomposable Families of Permutations. Whenever we ask that the circuit size parameter $s(N)$ is polynomial (in $\log(N)$), we do not know whether any efficiently computable (in both directions) permutation Π, Π^{-1} is decomposable. Intuitively, the reason is that while any permutation can be written as a concatenation of neighbor swaps, it may very well be the case that somewhere along the (likely exponentially-long) sequence of permutations, some of them will not have an efficient circuit implementation. This leads to our Question 8 regarding the ability to efficiently decompose permutations. Despite our lack of complete understanding of the decomposability of permutations, we mention some examples of decomposable families of permutations.

- **Naive composition of decomposable permutations.** Let Γ that can be decomposed into a polynomial-length sequence $\Gamma = \Gamma^1 \circ \cdots \circ \Gamma^r$ such that for every $i \in [r]$, the permutation Γ^i is (T, s)-decomposable. Then Γ is (rT, rs)-decomposable.
- **Linear chains.** Linear chains $(j \;\; j+1 \;\; j+2 \;\; \cdots \;\; \ell-1 \;\; \ell)$ swapping j and $j+1$, then $j+1$ and $j+2$ and eventually $\ell-1$ and ℓ. Equivalently, j goes to position ℓ and all other elements in the range are subtracted by 1. Linear chains as well as their inverses, are $(N, \mathsf{polylog}(N))$-decomposable. To see this one can consider a straightforward decomposition of the chain into neighbor swaps, and furthermore the efficient implementation of each of the intermediate permutations is given by a circuit that simply check if z is in the range of the chain, if so, decrements by 1 (mod N) or if the element is j, sends it to position ℓ.
- **Transpositions.** Transpositions $(j \;\; \ell)$, or (non-neighboring) swaps, are $(O(N), \mathsf{polylog}(N))$-decomposable using the decomposition

$$\begin{aligned}(j \;\; \ell) &= (j \;\; j+1 \;\; j+2 \;\; \cdots \;\; \ell-1)\,(\ell-1 \;\; \ell)\,(j \;\; j+1 \;\; j+2 \;\; \cdots \;\; \ell-1)^{-1} \\ &= (j \;\; j+1 \;\; j+2 \;\; \cdots \;\; \ell-1 \;\; \ell)\,(j \;\; j+1 \;\; j+2 \;\; \cdots \;\; \ell-1)^{-1} \;\;,\end{aligned}$$

  and then, to see the $(O(N), \mathsf{polylog}(N))$ decomposition, we use the above rule of composition of decomposable permutations, for the two decomposable permutations $(j \;\; j+1 \;\; j+2 \;\; \cdots \;\; \ell-1 \;\; \ell)$ and $(j \;\; j+1 \;\; j+2 \;\; \cdots \;\; \ell-1)^{-1}$.
- **Permutations that are decomposable to transpositions, rather than neighbor swaps.** Let Γ a permutation on $[N]$ that's (T, s)-decomposable, but to transpositions rather than neighbor swaps. Specifically, in the T-length sequence of permutations that Γ decomposes to (and each of such permutation Γ_i, Γ_i^{-1} has implementation of complexity $\leq s$), each consecutive pair is either identical or differs in one transposition. Then, Γ is $(T \cdot O(N), s + \mathsf{polylog}(N))$-decomposable into neighbor swaps.

- **Involutions.** Involutions are permutations where $\Gamma \circ \Gamma$ is the identity. Involutions that are computable by circuits of size s are $\big(O(N^2), s + \mathsf{polylog}(N)\big)$-decomposable. We will decompose Γ into a (N, s)-decomposition of transpositions, which will imply our wanted decomposition to neighbor swaps. Intuitively, we will visualize Γ as the applications of disjoint transpositions, and then the way we are going to decompose Γ is by adding each of the (possibly exponentially many) transpositions, only when both of its elements are smaller than some index. Formally,

$$\Gamma_i(x) := \begin{cases} \Gamma(x) & \text{if } x \le i \text{ and } \Gamma(x) \le i \\ x & \text{otherwise} \end{cases}.$$

  Each permutation in the sequence is of complexity $\le s + \mathsf{polylog}(N)$, we have Γ_0 is the identity and $\Gamma_N := \Gamma$. Also, $\Gamma_i = \Gamma_{i-1}$ if $\Gamma(i) \ge i$ and otherwise $\Gamma_i(x) = \Gamma_{i-1} \circ (i \;\; \Gamma(i))$.

- **Affine transformations.** Affine permutations $\mathbf{x} \mapsto \mathbf{A} \cdot \mathbf{x} + \mathbf{b} \bmod r$ where $\mathbf{A} \in \mathbb{Z}_r^{n \times n}$ is invertible and $\mathbf{b} \in \mathbb{Z}_r^n$, are $\big(O(r^n \times n^2), \mathsf{poly}(r, n)\big)$-decomposable. Note that the circuit size depends polynomially on r; we can improve this $\big(O(r^{2n} \times n^2), \mathsf{poly}(\log r, n)\big)$-decomposability under ERH. We leave it as an interesting open question to handle general r unconditionally. We observe that we can handle addition by $\mathbf{b}$ as above. To multiply by $\mathbf{A}$, we decompose $\mathbf{A}$ into $O(n^2)$ elementary row operations. Row Swaps are involutions and Row Sums are controlled additions, which are both decomposable by the above results. Finally, since r is small, we can handle Scalar Multiplications by the above.

- **Conjugations.** If $\Gamma = \Lambda^{-1} \circ \Gamma' \circ \Lambda$ where Λ, Λ^{-1} have circuits of size U and Γ' is (T, S)-decomposable, then Γ is $(T, O(S + U))$-decomposable, by simply conjugating the decomposition of Γ by Λ. Note that Λ does *not* need to be decomposable. As a particular example, the permutation $\mathbf{x} \mapsto \mathbf{x} + \mathbf{s} \bmod r$ for some fixed vector $\mathbf{s} \in \mathbb{Z}_r^n$ is $(O(r^n), \mathsf{polylog}(r^n))$-decomposable, since by conjugating with linear transformations we can turn it into $\mathbf{x} \mapsto \mathbf{x} + (1, 0, \cdots, 0) \bmod N$, which is reduces to the case of linear chains.

- **Scalar multiplication.** $x \mapsto ax \bmod N$ for any polynomial a which has an inverse in N is $(N, \mathsf{polylog}(N))$-decomposable, though this seems to require a bit of work. Under the Extended Riemann Hypothesis (ERH), all a are $(N^2, \mathsf{polylog}(N))$-decomposable. If discrete logarithms mod N had small circuits, we could use the conjugation example above to reduce multiplication to addition, decomposing multiplication by any a. However, since discrete logarithms are presumably classically hard, we have to do something else. In Remark 47 following the proof of Theorem 46, we explain how to decompose multiplications by small a, or more generally any $a \in \mathbb{Z}_N^*$ that is generated by small integers. Assuming ERH, all a are generated by small integers [Bac90], giving a conditional decomposability for all a.

- **Permutations with an ancilla.** We do not know how to generically decompose any permutation Γ, though we can do it if we are willing to use ancilla bits. In particular, for any permutation Γ on $[N]$ such that Γ, Γ^{-1} have size at most s, there exists a permutation Γ' on $[N]^2 \cong [N^2]$ that is $(O(N^4), s \times \mathsf{polylog}(N))$-decomposable, such that $\Gamma'(x||0) = \Gamma(x)||0$. Namely, let $\Gamma_0(x||y) = (x||\; y - \Gamma(x)\;)$, $\Gamma_1(x||y) = (x||\; -y\;)$, $\Gamma_2(x||z) = (\; x - \Gamma^{-1}(z))\;||z)$ and $\Gamma_3(w||z) = (z||w)$, and let $\Gamma' = \Gamma_3 \circ \Gamma_2 \circ \Gamma_1 \circ \Gamma_0$. Then we see that $x||0 \mapsto x||-\Gamma(x) \mapsto x||\Gamma(x) \mapsto 0||\Gamma(x) \mapsto \Gamma(x)||0$. Each of $\Gamma_0, \Gamma_1, \Gamma_2, \Gamma_3$ are involutions (since our arithmetic is

over $\mathbb{Z}_2$) on a domain $[N^2]$ and have size at most linear in the size of Γ, Γ^{-1}. Thus, they can be decomposed into a sequence of $O(N^4)$ permutations using the involution case.

- **Injective functions.** Using ancillas, we can even apply arbitrary (efficiently computable and invertible) injective functions. Namely, suppose $\Gamma : [N] \to [M]$ is an injective function such that Γ and Γ^{-1} are computable in size s. Then using the same construction as in permutations with ancillas, we can extend Γ into a permutation Γ' on $[N] \times [M] \cong [NM]$ which is $\left(O\left((NM)^2\right), s \times \mathsf{polylog}\left(NM\right)\right)$-decomposable.
- **Applying a decomposable permutation to a subset of the bits.** If $N = N_0 \cdot N_1$, and Γ_0 is a (T, s)-decomposable permutation on domain $[N_0]$, we can extend it to a permutation Γ with domain $[N] \cong [N_0] \times [N_1]$, that applies Γ_0 to $[N_0]$ and the identity to $[N_1]$. Then Γ is also (T, s)-decomposable.
- **Applying a conditional decomposable permutation.** If $N = N_0 \cdot N_1$, and Γ_0 is a (T, s)-decomposable permutation on domain $[N_0]$, we can extend it to a permutation Γ with domain $[N] \cong [N_0] \times [N_1]$ where Γ applies Γ_0 to $[N_0]$ conditioned on some target value $v \in [N_1]$, and the identity otherwise. Then Γ is $(T, s + \mathsf{polylog}(N))$-decomposable.
- **Applying a controlled decomposable permutation.** If $N = N_0 \cdot N_1$, and for every $v \in [N_1]$, Γ_v is a (T, s)-decomposable permutation on domain $[N_0]$, we can extend to a permutation Γ with domain $[N] \cong [N_0] \times [N_1]$ where Γ applies Γ_v to $[N_0]$ conditioned on the element in $[N_1]$ being. Then Γ is $(T \cdot N_1, s + \mathsf{polylog}(N))$-decomposable.

Our main theorem of this section is the following:

Theorem 39. *Let T be any exponential function and p any polynomial. Assuming the existence of sub-exponentially-secure one-way functions and sub-exponentially-secure iO, there exists an OP-PRP for the class of (T, p)-decomposable permutations. Moreover, the OP-PRP is itself (T, p) decomposable.*

This theorem will be proved in Sections 5.2, 5.3, and 5.4. Before proving it, however, we will give some example applications.

5.1 How to use OP-PRPs with Indisitnguishability Obfuscation

Here, we explain how OP-PRPs are useful for constructions involving indistinguishability obfuscation.

Composing with fixed permutations. Consider a program $P^{O,O^{-1}}$ which makes queries to an oracle O. We show the following:

Lemma 40. *Let Γ be a permutation, and let $(\Pi, \Pi^{-1}, \mathsf{Permute})$ be an OP-PRP for a class of permutations which includes Γ. Then for a sufficiently large polynomial s, $\mathsf{iO}(1^\lambda, 1^s, P^{\Pi(k,\cdot),\Pi^{-1}(k,\cdot)})$ is computationally indistinguishable from $\mathsf{iO}(1^\lambda, 1^s, P^{\Gamma(\Pi(k,\cdot)),\Pi^{-1}(k,\Gamma^{-1}(\cdot))})$, where $k \leftarrow \{0,1\}^\lambda$ is uniformly random.*

In other words, we can compose $\Pi(k, \cdot)$ with any fixed permutation Γ applied to the output of Π. This is analogous to the oracle case, where composing a random permutation with any fixed permutation gives a random permutation.

Proof. We prove security through a sequence of hybrids:

Hyb_0: Here, the adversary is given $\mathsf{iO}(1^\lambda, 1^s, P^{O,O^{-1}})$ where $O(\cdot) = \Pi(k, \cdot)$ and $O^{-1}(\cdot) = \Pi^{-1}(k, \cdot)$.

Hyb_1: Now we sample $k^{\Gamma,0} \leftarrow \mathsf{Permute}(k, \Gamma, 0)$ and switch to $O(\cdot) = \Pi(k^{\Gamma,0}, \cdot)$ and $O^{-1}(\cdot) = \Pi^{-1}(k^{\Gamma,0}, \cdot)$. By the correctness of the permuted key $k^{\Gamma,0}$, O, O^{-1}, and hence $P^{O,O^{-1}}$, is unchanged by this modification. Therefore, as long as s is larger than the maximum size of $P^{O,O^{-1}}$ in Hybrids 0 and 1, by iO security the two hybrids are indistinguishable.

Hyb_2: Now we switch to $k^{\Gamma,1} \leftarrow \mathsf{Permute}(k, \Gamma, 1)$ and set $O(\cdot) = \Pi(k^{\Gamma,1}, \cdot)$ and $O^{-1}(\cdot) = \Pi^{-1}(k^{\Gamma,1}, \cdot)$. Indistinguishability from Hybrid 1 follows from OP-PRP security.

Hyb_3: Now we move to $O(\cdot) = \Gamma(\Pi(k, \cdot))$ and $O^{-1}(\cdot) = \Pi^{-1}(k, \Gamma^{-1}(\cdot))$. Observe that these oracles O, O^{-1} are functionally identical to those in Hybrid 2, and therefore so is the program P. Thus, indistinguishability from Hybrid 2 follows from iO security.

Thus, we have that Hybrids 0 and 3 are indistinguishable, proving Lemma 40. □

Trapdoor permutations from iO and one-way functions. Here, we show that obfuscating an OP-PRP gives a trapdoor permutation.

Construction 41. *Let* $\left(\Pi, \Pi^{-1}\right)$ *be a PRP, and* iO *an indistinguishability obfuscator. Then define the trapdoor permutation* $\left(\mathsf{Gen}, F, F^{-1}\right)$ *as:*

- $\mathsf{Gen}\left(1^\lambda\right)$*: Sample* $k \leftarrow \{0,1\}^\lambda$*. Let* $P : \{0,1\}^\lambda \to \{0,1\}^\lambda$ *defined as* $P(x) := \Pi(k, x)$*, setting the block size* $n = \lambda$*. Let* s *be a sufficiently large function of* λ*. Output* $\mathsf{pk} = \hat{P} \leftarrow \mathsf{iO}\left(1^\lambda, 1^s, P\right)$ *and* $\mathsf{sk} = k$*.*
- $F\left(\mathsf{pk}, x\right)$*: Interpret* pk *as a program* $\hat{P}$*, and output* $\hat{P}\left(x\right)$*.*
- $F^{-1}\left(\mathsf{sk}, y\right)$*: Interpret* sk *as a key* k*, and output* $\Pi^{-1}\left(k, y\right)$*.*

Theorem 42 (Trapdoor One-Way Permutations from iO and One-Way Functions)**.** *Assume the existence of one-way functions. Assume* $\left(\Pi, \Pi^{-1}, \mathsf{Permute}\right)$ *is an OP-PRP for some class that includes all transpositions, and* iO *is a secure iO. Then for a sufficiently large polynomial* s*, if we instantiate Construction 41 with* $\left(\Pi, \Pi^{-1}, \mathsf{Permute}\right)$ *as the PRP, we get a secure trapdoor permutation.*

Proof. We need to show that there is no algorithm $\mathcal{A}$ which, given $\hat{P} \leftarrow \mathsf{iO}(1^\lambda, 1^s, P)$ and a random $y^* \leftarrow \{0,1\}^\lambda$, outputs x^* such that $P(x^*) = y^*$. Assume toward contradiction that there is such an $\mathcal{A}$ with success probability ϵ. We will show that ϵ is negligible through a sequence of hybrid experiments.

Hyb_0: Here, $\mathcal{A}$ is given $\hat{P} \leftarrow \mathsf{iO}(1^\lambda, 1^s, P)$ and y^*, where $y^* \leftarrow \{0,1\}^\lambda$ and $P(\cdot) = \Pi(k, \cdot)$ for a random key $k \leftarrow \{0,1\}^\lambda$. $\mathcal{A}$ wins if it outputs x such that $\Pi(k, x) = y^*$, which by assumption is with non-negligible probability.

Hyb_1: Now we additionally sample a random $y' \leftarrow \{0,1\}^\lambda$, and switch to $\hat{P} \leftarrow \mathsf{iO}(1^\lambda, 1^s, P_1)$ where

$$P_1(x) = \begin{cases} y^* & \text{if } \Pi(k, x)) = y' \\ \Pi(k, x) & \text{otherwise} \end{cases}$$

$\mathcal{A}$ still wins if it outputs x such that $\Pi(k, x) = y^*$. Indistinguishability from Hybrid 0 follows from the indistinguishability version of Lemma 17.

Hyb_2: Now we sample $k^* \leftarrow k^{(y^* \ y'),0} \leftarrow \mathsf{Permute}(k, (y^* \ y'), 0)$, change $\hat{P} \leftarrow \mathsf{iO}(1^\lambda, 1^s, P_2)$, where

$$P_2(x) = \begin{cases} y^* & \text{if } \Pi(k^*, x)) = y' \\ \Pi(k^*, x) & \text{otherwise} \end{cases}$$

Now we switch to $\mathcal{A}$ winning if it produces an x such that $\Pi(k^*, x) = y^*$. Since the programs $\Pi(k, \cdot)$ and $\Pi(k^{(y^* \ y'),0}, \cdot) := \Pi(k^*, \cdot)$ are functionally equivalent, the obfuscations of the programs P_1 and P_2 are indistinguishable by the security of the outer iO. Also, the success condition of the adversary is functionally equivalent. Overall, indistinguishability from Hybrid 1 follows from iO security.

Hyb_3: Now we switch to sampling $k^* = k^{(y^* \ y'),1} \leftarrow \mathsf{Permute}\,(k, (y^* \ y'), 1)$ and use this key in the program P_2 and also for the win condition. Indistinguishability from Hybrid 2 follows from OP-PRP security. Now observe that the win condition is $\Pi(k^{(y^* \ y'),1}, x) = y^*$, which is equivalent to $\Pi(k, x) = y'$.

Hyb_4: Now we switch back to giving $\mathcal{A}$ the program $\hat{P} \leftarrow \mathsf{iO}(1^\lambda, 1^s, P_1)$, but keeping the winning condition as $\Pi(k, x)) = y'$. Observe that switching from $k^* = k^{(y \ y'),0}$ to $k^* = k^{(y \ y'),1}$ in P_2 actually did not change the functionality at all: the change permuted the values of y^* and y' in the output of Π, but since both values cause P_2 to output y^*, swapping them does not change the functionality. Therefore, the functionality of P_2 using either permuted key remains equivalent to P_1. Thus, $\mathcal{A}$ outputs x such that $\Pi(k, x) = y'$ with non-negligible probability.

Now, observe that if we additionally give the adversary k, it can compute $y' = \Pi(k, x)$ for itself, with non-negligible probability. Thus, we obtain an adversary $\mathcal{B}$ which is given the description of P (namely, the key k) and $\mathsf{iO}(1^\lambda, 1^s, P_1)$, and guesses y' with non-negligible probability. But this contradicts the computational unpredictability guarantee from Lemma 17. Hence $\mathcal{A}$ original advantage in inverting the one-way permutation must be negligible. □

Fixed sparse triggers. Lemma 17 allows for "puncturing" a program if a certain random trigger is hit, in which case the program may behave completely differently from the original program. Here, we show that the trigger can even be *fixed*, as long as it is appropriately scrambled by OP-PRPs. Concretely, consider the program $P(x)$ with hardcoded OP-PRP keys k_0, k_1 that works as follows:

1. Apply some polynomial-sized circuit $P_0(x)$, obtaining a pair x_1, w_1.
2. Run $\Pi(k_0, x_1)$, and parse the output as (x_2, w_2).
3. Apply some polynomial-sized circuit $P_1(w_1, w_2)$, obtaining w_3, w_4.
4. Then run $\Pi^{-1}(k_1, \ (x_2, w_3)\ )$, obtaining x_3.
5. Finally feed x_3, w_4 into some polynomial-sized circuit P_2, and output the result.

The key structural property of the program P is that x_2 is the output of $\Pi(k_0, \cdot)$ and is fed into $\Pi^{-1}(k_1, \cdot)$ without modification and without affecting any other part of the cicuit.

Now consider a different program $P'(x)$, which is identical except that we modify Step 3 into Step 3' by embedding a trigger:

3'. Apply some polynomial-sized circuit $R(x_2)$ with single-bit outputs. If $R(x_2) = 0$, then let $(w_3, w_4) = P_1(w_1, w_2)$ as in program P. However, if $R(x_2) = 1$, we instead let $(w_3, w_4) = P_1'(w_1, w_2)$, for some different polynomial-sized circuit P_1'.

We will additionally consider the case where there are programs Q, Q' of the same form as P, P', but where the roles of k_0, k_1 are versed, and P_0, P_1, P_1', P_2 are replaced by arbitrary potentially different programs Q_0, Q_1, Q_1', Q_2. R will be same in both P' and Q'. We will show that obfuscating the un-primed and primed versions give computationally indistinguishable programs, for certain choices of R.

In the following, we will interpret x_2 as an element in $[0, N)$.

Lemma 43. *Suppose $R(x_2)$ outputs 1 if and only if $x_2 \in [a, b)$, for integers $0 \leq a < b < N$ such that $N/(b-a)$ is exponential in λ. Assuming $(\Pi, \Pi^{-1}, \mathsf{Permute})$ is an OP-PRP for any class that includes all involutions and iO is a secure iO, for random choices of the keys k_0, k_1, then for a sufficiently large polynomial s, we have that $\mathsf{iO}(1^\lambda, 1^s, P), \mathsf{iO}(1^\lambda, 1^s, Q)$ is computationally indistinguishable from $\mathsf{iO}(1^\lambda, 1^s, P'), \mathsf{iO}(1^\lambda, 1^s, Q)$.*

Proof. We prove indistinguishability through a sequence of hybrids.

Hyb_0: Here, we obfuscate P, Q.

Hyb_1: Here, we choose a random $y \in [0, \lceil N/(b-a) \rceil)$, and obfuscate the programs P', Q', but where we replace the relation R in both programs with the relation R' where $R'(x_2)$ outputs 1 if and only if $x_2 \in [y(b-a), (y+1)(b-a))$. This is the same as saying that $(x_2 - [x_2 \bmod (b-a)]) \;/\; (b-a) = y$. Observe that the range of $(x_2 - [x_2 \bmod (b-a)]) \;/\; (b-a)$ is contained in $\lceil N/(b-a) \rceil$. Indistinguishability of Hybrid 0 and Hybrid 1 follows from the indistinguishability guarantee of Lemma 17 and the fact that y is uniform in an exponential-sized domain . Hyb_2: Here, we switch to a random $y \in [0, \lfloor N/(b-a) \rfloor)$. Since $N/(b-a)$ is exponential, this is a negligible change in the distribution of y, hence Hybrid 1 and Hybrid 2 are indistinguishable.

Hyb_3: Let π be the involution on x_2, which exchanges the ranges $[a, b)$ and $[y(b-a), (y+1)(b-a))$. Notice that these intervals have the same size, and the latter interval is contained in $[0, N)$ since $y \leq N/(b-a) - 1$. We can easily extend π to be an involution mapping $(x_2, w_2) \mapsto (\pi(x_2), w_2)$ or $(x_2, w_3) \mapsto (\pi(x_2), w_3)$.

Now instead of obfuscating the program P', we switch to obfuscating the program P'' which replaces $\Pi(k_0, \cdot)$ with $\pi(\Pi(k_0, \cdot))$ and $\Pi^{-1}(k_1, \cdot)$ with $\Pi^{-1}(k_1, \pi^{-1}(\cdot))$ (still using the relation R'). We likewise switch from Q' to the analogous program Q''. Since this is just composing the PRP applications with the fixed involution π, Hybrids 1 and 2 are indistinguishable by Lemma 40.

Observe that in P'' we now we apply π to the output of $\Pi(k_0, \cdot)$ and to the input to $\Pi^{-1}(k_1, \cdot)$ (and the analogous statements for Q''). Thus, the two applications of π cancel out, *except* that the trigger is checked *between* applications of π. Since π exchanges the roles of $[a, b)$ and $[y(b-a), (y+1)(b-a))$, if were to test the output of $\Pi(k_0, \cdot)$ itself, the trigger value would in fact be the interval $[a, b)$.

Hyb_4: Now we switch to obfuscating P', Q' without π but with the correct relation R. This is functionally equivalent to our modified P'', Q'', since the permuted keys changes the trigger (when interpreted as an output of $\Pi(k_0, \cdot)$ or $\Pi(k_1, \cdot)$) to be $[a, b)$. Thus, by iO security, Hybrid 2 and Hybrid 3 are computationally indistinguishable. This completes the proof of Lemma 43. □

5.2 ONS-Merges

We now gradually build up to our proof of Theorem 39. Here, we start from a seemingly much weaker object called an Output Neighbor Swap Merge (ONS-Merge).

A neighbor swap is a permutation which exchanges some j with $j+1$ and otherwise is the identity. In chains notation, a neighbor swap would be written as $(j \;\; j+1)$.

A Merge is a permutation with the added correctness requirement. The domain $[N]$ is interpreted as pairs (b, x) for $b \in \{0,1\}$ and $x \in N_b$, where $N_0 + N_1 = N$. The range $[N]$ remains $[N]$. We let $L = \{(0,x)\} \cong [N_0]$ and $R = \{(1,x)\} \cong [N_1]$. A Merge is then a permutation preserves the ordering of elements in L, and also preserves the ordering of elements in R. An Output Neighbor Swap Merge can be thought of as an OP-PRP for the simple class of neighbor swaps. However, many such swaps will actually break the strong order-preserving property of the merge, and hence will be illegal. We therefore need to modify the definition (both correctness and security) to account for this.

Definition 44. *An Output Neighbor Swap (ONS-)* Merge *is at tuple of five algorithms* $(\mathsf{M}, \mathsf{M}^{-1}, \mathsf{Permute}, \mathsf{Eval}, \mathsf{Eval}^{-1})$ *with the following properties:*

- **Efficient Permutations:** *For any key $k \in \{0,1\}^\lambda$, any desired block-sizes N_0, N_1, $\mathsf{M}(k,\cdot)$ is an efficiently computable permutation on $[N = N_0 + N_1]$ with with $\mathsf{M}^{-1}(k,\cdot)$ being its efficiently computable inverse.*
- **Order-Preserving:** *For any key $k \in \{0,1\}^\lambda$, any block sizes N_0, N_1, and any two inputs $x_0 < x_1 \in [N_0]$ (resp. $x_0 < x_1 \in [N_1]$), then $\mathsf{M}(k,(0,x_0)) < \mathsf{M}(k,(0,x_1))$ (resp. $\mathsf{M}(k,(1,x_0)) < \mathsf{M}(k,(1,x_1))$). If $b_0 \neq b_1$, there is no restriction on the ordering of $\mathsf{M}(k,(b_0,x_0))$ and $\mathsf{M}(k,(b_1,x_1))$*
- **Output Neighbor Swapping:** $\mathsf{Permute}(k,\Gamma,c)$ *is a deterministic polynomial-time procedure which takes as input a key $k \in \{0,1\}^\lambda$, a neighbor swap $(z \;\; z+1)$, and a bit c. If $\mathsf{M}^{-1}(k,z) = (b_0,x_0)$ and $\mathsf{M}^{-1}(k,z+1) = (b_1,x_1)$ with $b_0 \neq b_1$, it outputs a swapped key $k^{(z\;\; z+1),c}$. Otherwise if $b_0 = b_1$, the input is considered illegal and the output is $\perp$.*
- **Output Swapping Correctness:** *For all $\lambda \in \mathbb{Z}$, $k \in \{0,1\}^\lambda$, all* legal *neighbor swaps $(z \;\; z+1)$, and all $x, z' \in [N]$,*

$$\mathsf{Eval}(k^{(z\;\; z+1),c}, x) = \begin{cases} \Pi(k,x) & \text{if } c = 0 \\ (z \;\; z+1) \circ (\Pi(k,x)) & \text{if } c = 1 \end{cases}$$

$$\mathsf{Eval}^{-1}(k^{(z\;\; z+1),b}, z') = \begin{cases} \Pi^{-1}(k,z') & \text{if } c = 0 \\ \Pi^{-1}(k,(z \;\; z+1)(z'))) & \text{if } c = 1 \end{cases}$$

- **Security:** *For any interactive QPT adversary $\mathcal{A}$, there exists a negligible function function $\epsilon(\lambda)$ such that the following experiment with $\mathcal{A}$ outputs 1 with probability at most $1/2 + \epsilon(\lambda)$:*
  - *$\mathcal{A}(1^\lambda)$ chooses a neighbor swap $(z \;\; z+1)$ for $z \in [N-1]$.*
  - *The experiment chooses a random $k \leftarrow \{0,1\}^\lambda$ and a random bit $c \in \{0,1\}$. It computes $(b_0,x_0) \leftarrow \mathsf{M}^{-1}(k,z)$ and $(b_1,x_1) \leftarrow \mathsf{M}^{-1}(k,z+1)$. It checks that $b_0 \neq b_1$; if $b_0 = b_1$ the experiment immediately aborts and returns a random bit. If $b_0 \neq b_1$, it returns $k^{(z\;\; z+1),b} \leftarrow \mathsf{Permute}(k,(z \;\; z+1),c)$ to $\mathcal{A}$.*

– *$\mathcal{A}$ produces a guess c' for c. The experiment outputs 1 if $c' = c$.*

Note that the abort condition is necessary: in these cases, permuting the output by $(z \;\; z+1)$ actually reverses the order of two strings belonging to the same set L or R. But this breaks the order-preserving property of the permuted key, which allows for easy distinguishing. However, if $\mathsf{M}^{-1}(k, z)$ and $\mathsf{M}^{-1}(k, z+1)$ have are in different sets L and R, then the order between them is arbitrary, and so we can hope that the permuted keys are indistinguishable.

Hypergeometric Distribution. Let $D_{N,t,s}$ the following distribution: let U be an arbitrary subset of $[N]$ of size s. Choose a random set V of $[N]$ of size exactly t, and output the number of elements in $S \cap U$. This distribution is known as the hypergeometric distribution and can be efficiently sampled. More specifically, there is a sequence of functions $D^{\kappa}_{N,t,s}$ with domain $\{0,1\}^{\kappa}$ such that $D^{\kappa}_{N,t,s}(r)$ for random coins r approximates a distribution that is $O(N \times 2^{-\kappa})$-close to $D_{N,t,s}$.

Tally Trees. In order to describe our construction, we introduce the notion of a tally tree. Consider a merge M. Assign to each range element z a bit b indicating the first bit of the pre-image of z. Thus, we obtain a sequence V of N bits, which determines the images of the sets L and R. The number of 0's is exactly N_0 and the number of 1's is exactly N_1. Observe that V is in bijection with the merge M. This is because once you choose the set of images of L (resp. R), the actual mapping from L (resp. R) to those images is fixed by the ordering.

Now, notice that V alone does not actually allow for *efficient* computation of M (nor M^{-1}), since to determine the image of, say, $(0, x)$, you would need to find the position in V of the x-th 0. But this presumably requires scanning exponentially-many bits in V to find the right location.

We can, however, speed this up by supplying more information, which is exactly the *tally tree*. A tally tree T is a binary tree with N leaves, one for each element of the range N. We will think of T as having a fixed topology that depends on N, with the goal of making T shallow. We will associate two quantities to each node z. The first is $s(z)$, which si the number of leaves of the subtree rooted at z. $s(z)$ is solely a function of the topology of T and will just be used for notational convenience.

The second value is $v(z)$, which is the total of all V_u values for all leaves u in the subtree rooted at z. Equivalently, $v(z) = V_z$ for all leaves, and $v(p)$ for any internal node is equal to the sum $v(p) = v(u) + v(w)$ where u, w are the left and right children of p. Observe that for the root ε, $v(\varepsilon) = N_1$.

Observe that we can equivalently sample a tally tree in reverse, starting from the root. We start by setting $v(\varepsilon) = N_1$. Then suppose we have set $v(p) = t$ for a node p with left child u and right child w. Setting $v(p) = t$ stipulates that among the $s(p)$ leaves of the tree rooted at p, t of them are set to 1. But as we haven't set any of the descendents of p yet, just the total of them, the distribution over the positions of those t 1's in the subtree rooted at p is uniformly random. We then have that $v(u)$is exactly distributed according to $D_{s(p),t,s(u)}$. We then set $v(w) = v(p) - v(u)$.

Notice that by sampling from appropriate hypergeometric distributions, we can sample the nodes of T in basically any order. For example, let C be a *cover* of T, meaning a set of nodes whose subtrees are disjoint and jointly include all of the leaves of T. We can sample $v(u)$ for all $u \in C$ in any order as follows. Let N' denote the portion of the domain yet to be determined (initially $N' = N$) and S the number of 1's remaining to allocate (initially $S = N_1$). Then in any order, we choose an element $u \in C$, sample $v(u) \leftarrow D_{N',s(u),S}$, and then update $N' \mapsto N' - s(u), S \mapsto S - v(u)$.

Notice that setting $v(u)$ for $u \in C$ determines $v(u')$ for all u' that are "above" the cover. Observe that through this process, the distribution of $v(u)$ for any $u \in C$ depends on the total $\sum_{u'} v(u')$ of all $v(u')$ sampled so far, but is otherwise independent of the actual values $v(u')$.

Given a tally tree T, we evaluate $(b, x) \leftarrow \mathsf{M}^{-1}(z)$ as follows. First set $b = V_z$ by looking the value stored at the leaf labeled z. Now by the ordering property of a merge, x is just a count of the number of $z' < z$ with $\mathsf{M}^{-1}(z)$ having the first bit b. We cannot directly count such z' in V (since there will be exponentially-many), but we can instead use the internal nodes of the tree T. Namely, let $U_L = \{u\}$ be the set of nodes that are *left* siblings of nodes on the path from root to z. Then $x = \sum_{u \in U_L} v(u)$. Observe that this process only visits a single path from root to leaf and its siblings, and therefore only $O(d)$ nodes where d is the depth of the tree.

To evaluate $\mathsf{M}(x)$ given T, we simply do a binary search, exploiting the ordered property and our ability to compute $\mathsf{M}^{-1}(x)$.

Our Construction. To give our construction, we will show how to implicitly generate a tally tree, which will then generate a merge. Rather than build the tally tree from the leafs toward the roots, our construction use the top-down generation, but pseudorandonly generate the values in the tree.

Construction 45. *Let $\mathsf{F} : \{0,1\}^\lambda \times \{0,1\}^* \to \{0,1\}^\kappa$, and suppose it has a puncturing algorithm Punc. Given a key $k \in \{0,1\}^\lambda$, the tally tree T is implicitly defined as follows. We deterministically choose a topology that has depth $O(\log N)$, which determines $s(z)$ for all nodes z. We set $v(\varepsilon) = N_1$. Then we recursively internal nodes as follows. For a node u that is a left child of some node p (which already has $s(p)$ defined), define $v(u) = D^\kappa_{s(p),v(p),s(u)}(\mathsf{F}(k,p))$. We then let $v(w)$ for the right child w to be $v(p) - v(u)$. Note that these values are not explicitly computed, but rather left implicitly determined by k.*

To permute a key k according to a legal swap $(z \;\; z+1)$ and bit c, collect into a set H all nodes in the paths from root to z or $z+1$, together with all the siblings of nodes on this path. Here, H stands for "hard-coded". Let P denote the nodes just on the paths, excluding $z, z+1$. Here, P stands for "punctured." Assume without loss of generality that $v(z) = 0$ and $v(z+1) = 1$. Let $\overline{k}^P \leftarrow \mathsf{Punc}(k, P)$. Then do the following:

- *If $c = 0$, output $k^{(z \;\; z+1),0} = (\overline{k}^P, \{(u, v(u))\}_{u \in H}$.*
- *If $c = 1$, output $k^{(z \;\; z+1),1} = (\overline{k}^P, \{(u, v'(u))\}_{u \in H}$ where*

$$v'(u) = \begin{cases} v(u) + 1 & \text{if } u \text{ is an ancestor of } z \text{ but not } z+1 \\ v(u) - 1 & \text{if } u \text{ is an ancestor of } z+1 \text{ but not } z \\ v(u) & \text{otherwise} \end{cases}$$

To evaluate $v(u)$ for some node u, we first look up if there is a pair $(u, v) \in H$, and if so produce the value v. Otherwise, we can use $\overline{k}^P$ to compute the remaining nodes.

Theorem 46. *Suppose $(\mathsf{F}, \mathsf{Punc})$ is a secure puncturable PRF. Assume $\kappa \geq \lambda + \log N$. Then the protocol given in Construction 45 is an ONS Merge. In particular, if $(\mathsf{F}, \mathsf{Punc})$ is ϵ-secure, then Construction 45 is $O(\epsilon + N^2 \times 2^{-\kappa})$-secure. Moreover, the permutations $\mathsf{M}(k, \cdot), \mathsf{M}^{-1}(k, \cdot)$ are $(O(N^2), \mathsf{polylog}(N))$-decomposable.*

Proof. We first argue decomposability. Consider a tally tree T. The "identity" tally tree T_0 simply has all the leaves of value 1 as the rightmost leaves. Define $T_{r,i}$ as the tree which has the first (left-most) $r-1$ 1's in the leaves in the correct position, the right-most $N_1 - r$ pushed all the way to the right, and the r-th 1 at position $N-(N_1-r)-i$. Let i_r be such that the $N-(N_1-r)-i_r$ is the correct position of the rth 1. The tally trees $T_{r,i}$ can all be succinctly represented: let C_r be the cover of the leaves containing the left-most $r-1$ 1's. Then $v(u)$ for u descendant from C_r is determined exactly as in T. Let C'_r be a cover for the $N_1 - r$ leaves that have all the right-most 1's. Then any u descendant from C'_r has $v(u) = s(u)$. Finally, let C''_r be a cover for the remaining nodes. Since there is at most 1 leaf descendant from C''_r that contains a 1, we can easily compute $v(u)$ for any u descendant from C''_r, by simply deciding whether that 1 is a descandant of u.

Then $T_0 = T_{1,0}$, $T_{r,i_r} = T_{r+1,0}$, and $T_{N_1+1,0} = T$. Going from $T_{r,i}$ to $T_{r,i+1}$ corresponds to a neighbor swap. This gives a sequence of length $O(N^2)$ connecting T_0 to T, showing that T is $(O(N^2), \mathsf{polylog}(N))$-decomposable.

We now prove security through a sequence of hybrids.

Hyb_0: Here, the adversary sees $k^{(z\ z+1),0} = (\overline{k}^S, \{(u, v_0(u))\}_{u\in S})$ where $v_0(u)$ is defined as $v_0(u) = v(u)$, which for the left child u of a parent p is equal to $D^{\kappa}_{s(p),v(p),s(u)}(\mathsf{F}(k,p))$. Observe that we only need to consider the left children, as the right children are determined by their parent and sibling. The hybrid outputs a random bit at rejects if $v_0(z) = v_0(z+1)$.

Hyb_1: Here, we replace $k^{(z\ z+1),0}$ with $(\overline{k}^S, \{(u, v_1(u))\}_{u\in S})$ where for left-children u, $v_1(u)$ is defined as $D^{\kappa}_{s(p),v(p),s(u)}(r_u)$ for independent random coins $r_u \leftarrow \{0,1\}^\kappa$. The hybrid outputs a random bit and rejects if $v_1(z) = v_1(z+1)$. By a straightforward reduction to punctured PRF security, Hybrid's 0 and 1 are indistinguishable except with probability ϵ.

Hyb_2: We now move to $(\overline{k}^S, \{(u, v_2(u))\}_{u\in S})$, where $v_2(u)$ is sampled as a fresh random sample from $D_{s(p),v(p),s(u)}$. The hybrid outputs a random bit and rejects if $v_2(z) = v_2(z+1)$. Each such sample is at most $O(N \times 2^\kappa)$-close to the distribution $D^{\kappa}_{s(p),v(p),s(u)}$ sampled in Hybrid 1, and there are at most $O(N)$ such samples. Therefore, Hybrids 1 and 2 are at most $O(N^2 \times 2^{-\kappa})$-close.

Hyb_3: Here, we define v_2 as in Hybrid 2, but give the adversary $(\overline{k}^S, \{(u, v_3(u))\}_{u\in S})$, where, assuming $v_2(z) = 0$ and $v_2(z+1) = 1$, we define

$$v'_2(u) = \begin{cases} v_2(u) + 1 & \text{if } u \text{ is an ancestor of } z \text{ but not } z+1 \\ v_2(u) - 1 & \text{if } u \text{ is an ancestor of } z+1 \text{ but not } z \\ v_2(u) & \text{otherwise} \end{cases}$$

We now argue that the views $\{(u, v_2(u))\}_{u\in S}$ and $\{(u, v'_2(u))\}_{u\in S}$ are actually distributed identically, even given $\overline{k}^S$. To do so, observe that an equivalent way to sample v_2 is as follows. Let C be the cover of T defined as $H \setminus P$, obtained by taking all the nodes in S that are *not* on the paths from root to z or $z+1$, and additionally include $z, z+1$ themselves. This is given by the red squares and yellow pentagons in Figure 4. Then sample $v_2(u)$ for $u \in C$ according to the algorithm for sampling covers of tally trees described above, starting with $z, z+1$ and moving to the rest of C. Then compute internal nodes "above" the cover by adding the values of the node's children, which gives all pairs $(u, v_2(u))$ for $u \in S$. The nodes "below" the cover C are then implicitly generated by F as before. The hybrid then rejects if $v_2(z) = v_2(z+1)$.

Since the values of T for the cover C is distributed exactly as in the case of a random merge, we see that conditioned on not rejecting, $v_2(z)$ and $v_2(z+1)$ are distributed as random distinct bits.

Hybrid 3 then is identical, except that we swap the values of at z and $z+1$. But since $v(z), v(z+1)$ were random distinct values anyway, this distributions are identical. Moreover, the rest of the cover C is sampled only depending on the total $v(z)+v(z+1)$, which we know is 1 in both Hybrid 3 and Hybrid 2. Thus, the distribution of the entire cover C, and therefore the entire tree, is identical in Hybrid 2 and Hybrid 3.

Hyb_4 and Hyb_5: These are analogs of Hybrids 1,0 (respectively), except that we use the values v'_1, v'_0, which are derived from v_1, v_0 analogous to v'_2. Hybrids 3 and 4 are indistinguishable by an identical argument to the indistinguishability of Hybrids 1 and 2. Hybrids 4 and 5 are likewise indistinguishable by an identical argument to that of Hybrids 0 and 1.

Then we observe that Hybrid 5 uses $v'_0 = v'$, and is therefore exactly the case $k^{(z\ z+1),1}$. This finishes the proof. □ □

Remark 47. *The decomposition of merges actually allows us to perform scalar multiplication for small scalars. We observe that $x \mapsto 2x \bmod N$ for odd N is actually a merge: indeed,it preserves the ordering for $x \in [0,(N-1)/2]$ as well as for $x \in [(N+1)/2, N-1]$. Moreover, we can compute the associated tally tree rather trivially. Thus we can decompose it as in Theorem 46. We can likewise handle $x \mapsto ax \bmod N$ for N relatively prime to a, as long as a is polynomial. The idea is to generalize the concept of a merge where the input domain is partitioned into a buckets and the order-preserving property holds for each bucket. We can then moreover generate $x \mapsto ax \bmod N$ for any a which can be decomposed as the product of "small" (polynomial in $\log N$) a'. Note that if the Extended Riemann Hypothesis is true, such small a' generate al a [Bac90].*

5.3 From ONS-Merge to ONS-PRP

An Output Neighbor Swappable PRP is an OP-PRP for the class of neighbor swaps$(z\ \ z+1)$. We now show that an ONS-Merge can be used to construct an ONS-PRP. The construction is based on the small-domain PRP of [GP07], though (1) we will use it in the large-domain setting, and (2) we will be implicitly implementing the underlying PRF with a *puncturable* PRF.

The construction can be thought of as performing a merge-sort in reverse, where we first partition the domain into a left and right set, of sizes N_0 and N_1 respectively. Then all the left elements are recursively shuffled using a smaller PRP, and all the right elements are shuffled using an independent smaller PRP. This step is the opposite of sorting the left and right parts separately. Then the two halves are randomly inserted into their final positions using a merge, which preserves the order of each half. The keys for the smaller PRPs and the merge will be derived pseudorandomly from the overall key.

Construction 48. *Let $(\mathsf{M}, \mathsf{M}^{-1}, \mathsf{Permute}')$ be an ONS-Merge. Let PRG be a length-tripling PRG. Then let $(\Pi, \Pi^{-1}, \mathsf{Permute})$ be defined as follows. For $N=2$ (x is a single bit), we will simply have $\Pi(k,x) = k \oplus x$. The only permutation is $(0\ \ 1)$, which is the same as $x \mapsto 1 \oplus x$. We can permute the key information-theoretically as $\mathsf{Permute}(k,(0\ \ 1)) = k \oplus 1$. For larger N, we let $N_0 = \lfloor N/2 \rfloor$ and $N_1 = N_0 - N_1$. We interpret the domain as $[N] \cong \{(b,x)\}_{b\in\{0,1\},x\in[N_b]}$. We use this interpretation both for the input to $\Pi(k,\cdot)$ as well as for the input to M. For the outputs of $\Pi(k,\cdot)$ and M, we will interpret the range as $[N]$. Then we define the algorithms as follows:*

- $\Pi(k,(b,x))$: *Run* $(k_0,k_1,k_2) \leftarrow \mathsf{PRG}(k)$, $y \leftarrow \Pi(k_b,x)$, *and output* $z \leftarrow \mathsf{M}(k_2,(b,y))$. *Here,* $\Pi(k_b,x)$ *is called recursively.*
- $\Pi^{-1}(k,z)$: *Run* $(k_0,k_1,k_2) \leftarrow \mathsf{PRG}(k)$, $(b,y) \leftarrow \mathsf{M}^{-1}(k_2,z)$ *and Compute* $x \leftarrow \Pi^{-1}(k_b,y)$ *and output* (b,x). *Here,* $\Pi^{-1}(k_b,y)$ *is called recursively.*
- $\mathsf{Permute}(k,(z\ \ z+1),c)$: *Let* $(k_0,k_1,k_2) \leftarrow \mathsf{PRG}(k)$. *Compute* $(b_0,x_0) \leftarrow \Pi^{-1}(k,z)$ *and* $(b_1,x_1) \leftarrow \Pi^{-1}(k,z+1)$. *We break into two cases:*
  - *If* $b_0 \neq b_1$ *output the permuted key* $k^{(z\ \ z+1),c} = (k_0,k_1,k_2^{(z\ \ z+1),c})$ *where* $k_2^{(z\ \ z+1),c} \leftarrow \mathsf{Permute}'(k_2,(z\ \ z+1),c)$.
  - *If* $b_0 = b_1 = b$, *output the permuted key* $k^{(z\ \ z+1),c} = (\overline{k_0},\overline{k_1},k_2)$ *where* $\overline{k_{1-b}} = k_{1-b}$ *and* $\overline{k_b} = k_b^{(y\ \ y+1),c} \leftarrow \mathsf{Permute}(k_b,(y\ \ y+1),c)$ *where* $(b,y) \leftarrow \mathsf{M}^{-1}(k_2,z)$. *Here,* $\mathsf{Permute}$ *is recursively called.*
- $\Pi(k^{(z\ \ z+1),c},x)$: *Run* $\Pi(k,x)$ *exactly as above but swapping out* k_0,k_1,k_2 *with their permuted versions if necessary. Likewise define* $\Pi^{-1}(k^{(z\ \ z+1),c},x)$.

Theorem 49. *If* PRG *is a secure PRG and* $(\mathsf{M},\mathsf{M}^{-1},\mathsf{Punc}')$ *is a secure ONS-Merge, then* $(\Pi,\Pi^{-1},\mathsf{Punc})$ *in Construction 45 is a secure ONS-PRP. In particular, if* PRG *is* ϵ_{PRG}*-secure and* $(\mathsf{M},\mathsf{M}^{-1},\mathsf{Punc}')$ *is* ϵ_{M}*-secure, then* $(\Pi,\Pi^{-1},\mathsf{Punc})$ *is* ϵ_Π*-secure where* $\epsilon_\Pi(\lambda,n) \leq O((\epsilon_{\mathsf{PRG}}(\lambda)+\epsilon_{\mathsf{M}}(\lambda,N)) \times \log N)$. *Moreover, the permutations* $\Pi(k,\cdot)$ *and* $\Pi^{-1}(k,\cdot)$ *are* $(O(N^4),\mathsf{polylog}(N))$*-decomposable.*

Proof. First, we argue efficiency. A call to Π on block-size N makes a call to PRG, a call to M on block-size N, and a recursive call to Π on block-size at most $\lceil N/2 \rceil$. Solving the recurrence, Π makes $O(\log N)$ calls to PRG and $O(\log N)$ calls to M on block-sizes no more than N. Since PRG nad M are polynomial time, this is efficient. We can likewise analyze the other algorithms.

Next we need to argue the correctness guarantees. In particular, we need to argue that $\mathsf{Permute}$ makes called to $\mathsf{Permute}'$ on neighbor swaps where the pre-images of the swapped points have different initial bits. Moreover, we need to verify that Π using a punctured key computes the correct function.

For the case $n=1$, Π is just XORing with the key k. the only available neighbor swap is $(0\ \ 1)$, which is the same as XORing with 1, which is equivalent to XORing the key with 1. Thus, the $n=1$ case achieves perfect security. Now we handle the $n>1$ case.

Consider running $\mathsf{Permute}(k,(z\ \ z+1),c)$. Recall that $\mathsf{Permute}'(k_2,(z\ \ z+1),c)$ is called exactly when the first bits of $\Pi^{-1}(k,z)$ and $\Pi^{-1}(k,z+1)$ are different. Notice that this is equivalent to the case where $\mathsf{M}^{-1}(k_2,z)$ and $\mathsf{M}^{-1}(k_2,z+1)$ have different first bits, which means that this is a valid call to $\mathsf{Permute}'$. In this case, the correctness of the permuted key follows immediately from the correctness of the underlying M.

Now consider the case where $\Pi^{-1}(k,z)$ and $\Pi^{-1}(k,z+1)$ (or equivalently, $\mathsf{M}^{-1}(k_2,z)$ and $\mathsf{M}^{-1}(k_2,z+1)$) share the same first bit. This implies that $\mathsf{M}^{-1}(k_2,z)$ and $\mathsf{M}^{-1}(k_2,z+1)$ are adjacent (any supposed string between them would share the same first bit, and by order preserving, its image under $\mathsf{M}(k_2,\cdot)$ would need to lie between $z,z+1$, which is impossible). Thus, when $\mathsf{Permute}(k_b,(y\ \ y+1),c)$ is recursively called where $(b,y) \leftarrow \mathsf{M}^{-1}(k_2,z)$, we have that $(b,y+1) = \mathsf{M}^{-1}(k_2,z+1)$. Thus, the neighbor swap $(y\ \ y+1)$ is actually transposing $\mathsf{M}^{-1}(k_2,z)$ and $\mathsf{M}^{-1}(k_2,z+1)$. From this, correctness of the permuted key following immediately from the (inductively justified) correctness of the underlying Π for block-size $N/2$.

Now we argue decomposability. To decompose, we simply decompose $\mathsf{M}(k_2, \cdot)$, and then recursively decompose $\Pi(k_0, \cdot)$ and $\Pi(k_1, \cdot)$. Solving the recurrence gives the decomposition statement in Theorem 49.

Finally, we prove security. Let $\mathcal{A}$ be a supposed adversary for $(\Pi, \Pi^{-1}, \mathsf{Permute})$ with winning probability $1/2+\epsilon_\Pi(\lambda, n)$. Let $k \in \{0,1\}^\lambda$ denote the experiments random key k, and let $k_0, k_1, k_2 \leftarrow \mathsf{PRG}(k)$ denote the keys used to evaluate $\Pi, \Pi^{-1}, \mathsf{Punc}$, and let z_0, z_1 denote $\mathcal{A}'s$ chosen pair. We consider the following sequence of hybrids:

Hyb_0: This is the ONS-PRP security experiment. By assumption, the winning probability is at least $1/2 + \epsilon_\Pi(\lambda, N)$.

Hyb_1: This is the same experiment, except we switch to k_0, k_1, k_2 being uniformly random keys, and using them in all algorithms. By a straightforward reduction to the PRG security of PRG, we have that $\mathcal{A}$ still wins with probability at least $1/2 + \epsilon_\Pi(\lambda, N) - \epsilon_{\mathsf{PRG}}(\lambda)$.

Hyb_2: This is the same as Hybrid 1, except that we change the win condition. Namely, if the punctured key $k^{\Gamma,c}$ contains a punctured k_2, then the experiment aborts and outputs a random bit, independent of $\mathcal{A}$. By a straightforward reduction to the ONS-Merge security of $(\mathsf{M}, \mathsf{M}, \mathsf{Punc}')$, we have that $\mathcal{A}$ still wins with probability at least $1/2 + \epsilon_\Pi(\lambda, N) - \epsilon_{\mathsf{PRG}}(\lambda) - \epsilon_{\mathsf{M}}(\lambda, N)$.

Hyb_3: This is the same as Hybrid 1, except that we further change the win condition to always output a random bit. The only difference from Hybrid 2 is that when $k^{\Gamma,c}$ contains a punctured k_b, in Hybrid 2 the experiment outputs a bit dependent on $\mathcal{A}$'s output, whereas in Hybrid 3 it outputs a random bit. By a straightforward reduction to the ONS-Security of $(\Pi, \Pi^{-1}, \mathsf{Punc})$ with block-size at most $\lceil N/2 \rceil$, we have that $\mathcal{A}$ still wins with probability at least $1/2 + \epsilon_\Pi(\lambda, N) - \epsilon_{\mathsf{PRG}}(\lambda) - \epsilon_{\mathsf{M}}(\lambda, N) - \epsilon_\Pi(\lambda, \lceil N/2 \rceil)$. But observe that in Hybrid 3, the winning probability is exactly $1/2$. Thus we have that

$$\epsilon_\Pi(\lambda, N) \leq \epsilon_{\mathsf{PRG}}(\lambda) + \epsilon_{\mathsf{M}}(\lambda, N) + \epsilon_\Pi(\lambda, \lceil N/2 \rceil)$$

For the base case, observe that when $N = 2$, Π is a random permutation and the punctured key contains no information, so we trivially have $\epsilon_\Pi(\lambda, 2) = 0$. Solving the recurrence gives the statement of the theorem. □

5.4 Achieving OP-PRPs for Decomposable Permutations Using Obfuscation

Construction 50. *Let $(\Pi, \Pi^{-1}, \mathsf{Permute}')$ be an ONS-PRP (that is, an OP-PRP for the family of neighbor swaps $(z \;\; z+1)$). Let $G = (G_n)_n$ be a family of efficiently computable permutations, and s a parameter. Then let $(\Pi, \Pi^{-1}, \mathsf{Permute})$ be a new PRP where* Permute *does the following:*

- $\mathsf{Permute}(k, \Gamma, b)$*: Let*

$$P = \begin{cases} \mathsf{iO}(1^\lambda, 1^s, \Pi(k, \cdot)) & \text{if } b = 0 \\ \mathsf{iO}(1^\lambda, 1^s, \Gamma(\Pi(k, \cdot))) & \text{if } b = 1 \end{cases}$$

$$P^{-1} = \begin{cases} \mathsf{iO}(1^\lambda, 1^s, \Pi^{-1}(k, \cdot)) & \text{if } b = 0 \\ \mathsf{iO}(1^\lambda, 1^s, \Pi(k, \Gamma^{-1}(\cdot))) & \text{if } b = 1 \end{cases}$$

  Output $k^{\Gamma,b} = (P, P^{-1})$.

Then we augment Π, Π^{-1} so that $\Pi(\,(P, P^{-1})\,, x) = P(x)$ and $\Pi^{-1}(\,(P, P^{-1})\,, z) = P^{-1}(z)$.

Observe that this construction preserves the decomposability of Π, Π^{-1}, since the algorithms are exactly the same.

Theorem 51. *Suppose* iO *is* ϵ*-secure and* $(\Pi, \Pi^{-1}, \mathsf{Permute}')$ *is* δ*-secure as an ONS-PRP. Suppose* $G = (G_n)_n$ *is* (T, s')*-decomposable. Let* t *be the maximum circuit size of* Π, Π^{-1} *after hardcoding the key (including permuted keys from* $\mathsf{Permute}'$*). Then as long as* $s \geq s' + t$*,* $(\Pi, \Pi^{-1}, \mathsf{Permute})$ *is* $O(T(\epsilon + \delta))$*-secure as an OP-PRP for* G*.*

Proof. Let $\Gamma_0, \cdots, \Gamma_T = \Gamma$ be the sequence of permutations where $\Gamma_i = \Gamma_{i-1} \circ (z_i \;\; z_i + 1)$ and Γ_0 is the identity. Our goal is to show that the pair $\mathsf{iO}(1^\lambda, \Pi(k, \cdot)), \mathsf{iO}(1^\lambda, \Pi^{-1}(k, \cdot))$ is indistinguishable from $\mathsf{iO}(1^\lambda, \Gamma(\Pi(k, \cdot))), \mathsf{iO}(1^\lambda, \Pi^{-1}(k, \Gamma^{-1}(\cdot)))$. Do do so, we will introduce a sequence of hybrids:

Hyb_i: $\mathsf{iO}(1^\lambda, 1^s \Gamma_i(\Pi(k, \cdot))), \mathsf{iO}(1^\lambda, \Pi^{-1}(k, \Gamma_i^{-1}(\cdot)))$. In particular, Hybrid 0 is the case where $\mathsf{iO}(1^\lambda, \Pi(k, \cdot))$, $\mathsf{iO}(1^\lambda, \Pi^{-1}(k, \cdot))$ and Hybrid T is the case $\mathsf{iO}(1^\lambda, \Gamma(\Pi(k, \cdot)))$, $\mathsf{iO}(1^\lambda, \Pi^{-1}(k, \Gamma^{-1}(\cdot)))$.

$\mathsf{Hyb}_{(i-1).1}$: $\mathsf{iO}(1^\lambda, , 1^s, \Gamma_i(\Pi(k^{\{(z_i \;\; z_i+1),0\}}, \cdot))), \mathsf{iO}(1^\lambda, , 1^s, \Pi^{-1}(k^{\{(z_i \;\; z_i+1),0\}}, \Gamma_i^{-1}(\cdot)))$. Here, $k^{\{(z_i \;\; z_i+1),0\}} \leftarrow \mathsf{Permute}'(k, (z_i \;\; z_i + 1), 0)$.

$\mathsf{Hyb}_{(i-1),2}$: $\mathsf{iO}(1^\lambda, 1^s, \Gamma_i(\Pi(k^{(z_i \;\; z_i+1),1}, \cdot))), \mathsf{iO}(1^\lambda, 1^s, \Pi^{-1}(k^{(z_i \;\; z_i+1),1}, \Gamma_i^{-1}(\cdot)))$. Here, $k^{(z_i \;\; z_i+1),1} \leftarrow \mathsf{Permute}'(k, (z_i \;\; z_i + 1), 1)$.

Observe that $\Pi(k^{(z_i \;\; z_i+1),0}, \cdot)$ is functionally equivalent to $\Pi(k, \cdot))$, and this equivalence is preserved by composing both sides with Γ_{i-1}. Likewise for the Π^{-1} functions. The sizes of the programs being obfuscated are at most $s' + t$, where t is the maximum size of the circuit Π after hardcoding the key (inclunding permuted keys by $\mathsf{Permute}'$). Thus, by iO security, Hybrids $i - 1$ and $(i - 1).1$ are indistinguishable, except with probability at most ϵ.

The only difference between Hybrids $(i - 1).1$ and $(i - 1).2$ is that we switch from $k^{(z_i \;\; z_i+1),0}$ to $k^{(z_i \;\; z_i+1),1}$. By the ONS-security of $(\Pi, \Pi^{-1}, \mathsf{Permute}')$, these two hybrids are indistinguishable except with probability at most δ.

Finally, observe that $\Pi(k^{(z_i \;\; z_i+1),1}, \cdot))$ is functionally equivalent to $(z_i \;\; z_i + 1) \circ \Pi(k, \cdot)$. Therefore $\Gamma_{i-1} \circ \Pi(k^{(z_i \;\; z_i+1),1}, \cdot))$ is functionally equivalent to $\Gamma_{i-1} \circ (z_i \;\; z_i + 1) \circ \Pi(k, \cdot)$, which in turn is equivalent to $\Gamma_i \circ \Pi(k, \cdot)$. Thus, by iO security, Hybrids $(i - 1).2$ and i are indistinguishable except with probability at most ϵ.

Thus we obtain a chain of hybrids $0 \to 0.1 \to 0.2 \to 1 \to 1.1 \to 1.2 \to 2 \to \cdots \to T - 1 \to (T - 1).1 \to (T - 1).2 \to T$ where each transition is indistinguishable. The triangle inequality then gives the theorem. □

6 One-Shot Signatures in the Standard Model

Following our construction in an oracle model from Section 4, we present our standard model construction of OSS.

Construction 52. *Let* $\lambda \in \mathbb{N}$ *the statistical security parameter. Define* $s := 16 \cdot \lambda$ *and let* $n, r, k \in \mathbb{N}$ *such that* $r := s \cdot (\lambda - 1)$*,* $n := r + \frac{3}{2} \cdot s$*,* $k := n$*. Let* $d := \mathsf{poly}_d(\lambda) \in \mathbb{N}$ *the expansion parameter and* $\kappa := \mathsf{poly}_\kappa(\lambda) \in \mathbb{N}$ *the cryptographic security parameter, for some sufficiently large polynomials in the statistical security parameter.*

Let iO *an iO scheme,* $(\mathsf{F}, \mathsf{Punc})$ *a puncturable PRF, and* $\big(\Pi, \Pi^{-1}, \mathsf{Permute}\big)$ *a permutable PRP for the class of all* $\big(2^{\mathsf{poly}(\lambda)}, \mathsf{poly}(\lambda)\big)$*-decomposable permutations. Then we construct a hash function* $(\mathsf{Gen}, \mathsf{Hash})$ *as follows:*

- $\mathsf{Gen}\left(1^\lambda\right)$: *Sample* $k_{\mathsf{in}}, k_{\mathsf{out}}, k_{\mathsf{lin}} \leftarrow \{0,1\}^\kappa$. $\Pi\left(k_{\mathsf{in}}, \cdot\right)$ *is a permutation with domain* $\{0,1\}^n$, $\Pi\left(k_{\mathsf{out}}, \cdot\right)$ *is a permutation with domain* $\{0,1\}^d$, *and* $\mathsf{F}\left(k_{\mathsf{lin}}, \cdot\right)$ *is a PRF with inputs in* $\{0,1\}^d$ *that outputs some polynomial number of bits. Let* $H(\cdot)$ *denote the first* r *output bits of* $\Pi\left(k_{\mathsf{in}}, \cdot\right)$ *and* $J(\cdot)$ *denote the remaining* $n-r$ *bits. For each* $y \in \{0,1\}^d$, *let* $\mathbf{A}(y) \in \mathbb{Z}_2^{k \times (n-r)}$ *be a matrix with full column-rank and* $\mathbf{b}(y) \in \mathbb{Z}_2^k$ *a vector, both are generated pseudorandomly by the output of* $\mathsf{F}\left(k_{\mathsf{lin}}, y\right)$.

  As the common reference string output $\mathsf{CRS} = \left(\mathcal{P}, \mathcal{P}^{-1}, \mathcal{D}\right)$ *where* $\mathcal{P} \leftarrow \mathsf{iO}\left(1^\kappa, P\right)$, $\mathcal{P}^{-1} \leftarrow \mathsf{iO}\left(1^\kappa, P^{-1}\right)$, $\mathcal{D} \leftarrow \mathsf{iO}\left(1^\kappa, D\right)$ *such that*

$$P(x) = \left(y \ , \ \mathbf{A}(y) \cdot J(x) + \mathbf{b}(y) \right) \text{ where } y \leftarrow \Pi^{-1}\left(k_{\mathsf{out}}, \ H(x) || 0^{d-r}\right)$$

$$P^{-1}\left(y, \mathbf{u}\right) = \begin{cases} \Pi^{-1}\left(w || \mathbf{z}\right) & \exists\, w, \mathbf{z} : \left(\Pi\left(k_{\mathsf{out}}, y\right) = w || 0^{d-r}\right) \wedge \left(\mathbf{A}(y) \cdot \mathbf{z} + \mathbf{b}(y) = \mathbf{u}\right) \\ \bot & \text{else} \end{cases}$$

$$D\left(y, \mathbf{v}\right) = \begin{cases} 1 & \text{if } \mathbf{v}^T \cdot \mathbf{A}(y) = 0^{n-r} \\ 0 & \text{otherwise} \end{cases}$$

- $\mathsf{Hash}\left(\mathsf{CRS}, x\right)$: *Compute* $(y, \mathbf{u}) \leftarrow \mathcal{P}(x)$ *and output* y.

The main differences between the oracle model Construction 27 and the standard model Construction 52 are that (1) Construction 52 pseudorandomly generates the permutations Π and matrices $\mathbf{A}_y, \mathbf{b}_y$, (2) that it obfuscates the programs rather than putting them in oracles, and (3) it additionally pads $H(x)$ with 0's and then applies $\Pi^{-1}(k_{\mathsf{out}}, \cdot)$ in order to get y, rather than setting $y = H(x)$.

The rationale behind expanding y (the input to $\mathsf{F}\left(k_{\mathsf{lin}}\right)$) is that in the security proof, we are going to use permutable PRPs together with trapdoor functions in a number of ways, and will need the expansion to be able to compose the functions with the PRPs in our scheme. Note that our choice to use the inverse $\Pi^{-1}\left(k_{\mathsf{out}}, \cdot\right)$ in P may seem arbitrary. However, our choice corresponds to us ultimately using $\Pi\left(k_{\mathsf{out}}, \cdot\right)$ as an output-permutable permutation, but we need to be applying the output permutations to the *input*. We accomplish this exactly by using the inverse of Π. While in Section 5 we show that output-permutable and input-permutable PRPs are in fact equivalent, this gives insight to the security proof.

Non-collapsing. The non-collapsing property of our standard model construction is identical to the oracle model non-collapsing procedure, shown in Proposition 28.

Security in the Standard Model. The rest of this section is for dedicated for showing security in the standard model. The results in Sections 6.1 (Lemma 54), 6.2 (Lemma 55) and 6.3 (Theorem 56) together, eventually prove the below theorem.

Theorem 53 (Collision Resistance of Hash)**.** *Let* $\mathsf{Gen}\left(1^\lambda\right)$ *the generation algorithm from Construction 52. Then, for every quantum polynomial-time algorithm* $\mathcal{A}$ *there exists a negligible function* negl *such that,*

$$\Pr\left[(x_0 \neq x_1) \wedge \left(\mathsf{Hash}(x_0) = \mathsf{Hash}(x_1)\right) \ : \ \begin{matrix} \left(\mathcal{P}, \mathcal{P}^{-1}, \mathcal{D}\right) & \leftarrow \mathsf{Gen}\left(1^\lambda\right) \\ \left(x_0, x_1\right) & \leftarrow \mathcal{A}\left(\mathcal{P}, \mathcal{P}^{-1}, \mathcal{D}\right) \end{matrix}\right] \leq \mathsf{negl}\left(\lambda\right) \ .$$

6.1 Bloating the Dual

We define the modified generator $\widetilde{\mathsf{Gen}}(1^\lambda, n, r, k, s)$, as follows. It samples a distribution over $\mathcal{P}, \mathcal{P}^{-1}, \mathcal{D}'$, where n, r, k correspond to their exact definitions in the original construction. For s we let $\mathbf{A}^{(0)}(y) \in \mathbb{Z}_2^{k \times s}$ denote the first s columns of $\mathbf{A}(y) \in \mathbb{Z}_2^{k \times (n-r)}$ and $\mathbf{A}^{(1)}(y) \in \mathbb{Z}_2^{k \times (n-r-s)}$ denote the remaining $n-r-s$ columns. Note that the standard generator is defined as $\widetilde{\mathsf{Gen}}(1^\lambda, n, r, k, 0)$. The functionality of $\mathcal{P}, \mathcal{P}^{-1}$ stays the same and we define $\mathcal{D}'(y, \mathbf{v})$ as an indistinguishability obfuscation of the following function:

$$D'(y, \mathbf{v}) = \begin{cases} 1 & \text{if } \mathbf{v}^T \cdot \mathbf{A}^{(1)}(y) = 0^{n-r-s} \\ 0 & \text{otherwise} \end{cases}$$

Observe that if $\mathbf{v}^T \cdot \mathbf{A}(y) = 0^{n-r}$, then $\mathbf{v}^T \cdot \mathbf{A}^{(0)}(y) = 0^{n-r-s}$. Thus $\mathcal{D}'$ accepts all points that are accepted by $\mathcal{D}$, but also accepts additional points as well, namely those for which $\mathbf{v}^T \cdot \mathbf{A}^{(0)}(y) \neq 0^s$ but $\mathbf{v}^T \cdot \mathbf{A}_y^{(1)} = 0^{n-r-s}$. We call this bloating the dual.

Lemma 54. *Let $\lambda \in \mathbb{N}$, and assume there is a quantum algorithm $\mathcal{A}$ with complexity $T_\mathcal{A}$ such that,*

$$\Pr\left[(y_0 = y_1 := y) \wedge (x_0 \neq x_1) \ : \ \begin{array}{rl} (\mathcal{P}, \mathcal{P}^{-1}, \mathcal{D}) & \leftarrow \widetilde{\mathsf{Gen}}\left(1^\lambda, n, r, k, 0\right) \\ (x_0, x_1) & \leftarrow \mathcal{A}\left(\mathcal{P}, \mathcal{P}^{-1}, \mathcal{D}\right) \\ (y_b, \mathbf{u}_b) & \leftarrow \mathcal{P}\left(x_b\right) \end{array}\right] \geq \epsilon \ .$$

Assume that the primitives used in Construction 52 are $\left(f(\cdot), \frac{1}{f(\cdot)}\right)$-secure for some sub-exponential $f(\lambda) := 2^{\lambda^\delta}$ for some constant real number $\delta > 0$. Also, for $w := \lambda^{\frac{\delta}{2}}$, $s' := s - (n-r-s)$, assume all of the following:

1. $\frac{2^r \cdot \frac{k^2}{\epsilon}}{f(\kappa)} \leq o(1)$,
2. $\frac{\frac{k^2}{\epsilon}}{f(w)} \leq o(1)$,
3. $\frac{\frac{2^w}{\epsilon}}{f(n-r-s)} \leq o(1)$,
4. $\frac{(n-r-s) \cdot \frac{2^w}{\epsilon}}{2^{s'}} \leq o(1)$, *and*
5. $\frac{\frac{2^w}{\epsilon} \cdot \left(k^5 + \mathsf{poly}(n-r-s) + T_\mathcal{A}\right)}{f(n-r-s)} \leq o(1)$.

Then, it follows that,

$$\Pr\left[\begin{array}{l} y_0 = y_1 := y, \\ (\mathbf{u}_0 - \mathbf{u}_1) \notin \mathsf{ColSpan}\left(\mathbf{A}^{(1)}(y)\right) \end{array} \ : \ \begin{array}{rl} (\mathcal{P}, \mathcal{P}^{-1}, \mathcal{D}') & \leftarrow \widetilde{\mathsf{Gen}}(1^\lambda, n, r, k, s) \\ (x_0, x_1) & \leftarrow \mathcal{A}\left(\mathcal{P}, \mathcal{P}^{-1}, \mathcal{D}'\right) \\ (y_b, \mathbf{u}_b) & \leftarrow \mathcal{P}(x_b) \end{array}\right] \geq \frac{\epsilon}{512 \cdot k^2} \ .$$

Note that $\mathbf{u}_0 - \mathbf{u}_1 \notin \mathsf{ColSpan}\left(\mathbf{A}^{(1)}(y)\right)$ means in particular that $\mathbf{u}_0, \mathbf{u}_1$, and hence x_0, x_1, are distinct. Thus, the second expression means that $\mathcal{A}$ is finding collisions, but these collisions satisfy an even stronger requirement.

Proof. Let $\lambda \in \mathbb{N}$ and assume there is a $T_{\mathcal{A}}$-complexity algorithm $\mathcal{A}$ and a probability ϵ such that $\mathcal{A}$ gets $\left(\mathcal{P}, \mathcal{P}^{-1}, \mathcal{D}\right) \leftarrow \mathsf{Gen}\left(1^{\lambda}\right)$ and outputs a pair (x_0, x_1) of n-bit strings such that with probability ϵ we have $x_0 \neq x_1$ and $y_0 = y_1$. We next define a sequence of hybrid experiments. Each hybrid defines a computational process, an output of the process and a predicate computed on the process output. The predicate defines whether the (hybrid) process execution was successful or not.

- Hyb_0: The original execution of $\mathcal{A}$.

The process Hyb_0 is the above execution of $\mathcal{A}$ on input a sample from $\left(\mathcal{P}, \mathcal{P}^{-1}, \mathcal{D}\right) \leftarrow \widetilde{\mathsf{Gen}}\left(1^{\lambda}, n, r, k, 0\right)$. We define the output of the process as (x_0, x_1) and the process execution is considered as successful if x_0, x_1 are both distinct and collide in their y values. By definition, the success probability of Hyb_0 is ϵ.

- Hyb_1: Preparing to switch to a bounded number of cosets $(\mathbf{A}(y), \mathbf{b}(y))$, by using an obfuscated puncturable PRF argument and injective mode of a lossy function.

Let $(\mathsf{LF.KeyGen}, \mathsf{LF.F})$ a $\left(f(\cdot), \frac{1}{f(\cdot)}\right)$-secure lossy function scheme (as in Definition 16). Set $w := \lambda^{\frac{\delta}{2}}$ where $f(\lambda) := 2^{\lambda^{\delta}}$. Sample $\mathsf{pk}_{\mathsf{LF}} \leftarrow \mathsf{LF.KeyGen}\left(1^d, 0, 1^w\right)$ and let $\mathsf{LF.F}\left(\mathsf{pk}_{\mathsf{LF}}, \cdot\right) : \{0,1\}^d \rightarrow \{0,1\}^m$ the induced injective function.

We now consider two circuits in order to describe our current hybrid. $E_0\left(k_{\mathsf{lin}}, \cdot\right)$ is the circuit that given an input from $\{0,1\}^d$ applies $\mathsf{F}\left(k_{\mathsf{lin}}, \cdot\right)$ to get $\mathbf{A}(y)$, $\mathbf{b}(y)$. $E_1\left(\mathsf{pk}_{\mathsf{LF}}, k'_{\mathsf{lin}}, \cdot\right)$ is the circuit that for a LF key $\mathsf{pk}_{\mathsf{LF}}$ and P-PRF key k'_{lin} for a P-PRF with input size m rather than d, given a d-bit input y, applies the lossy function with key $\mathsf{pk}_{\mathsf{LF}}$ and then the P-PRF to get $\mathbf{A}'(y)$, $\mathbf{b}'(y)$.

Note that in the previous hybrid, the circuit E_0 is used in all three circuits P, P^{-1}, D in order to generate the cosets per input y, and furthermore, each of these three circuits access E_0 only as a black box. The change that we make to the current hybrid is that we are going to use $E_1\left(\mathsf{pk}_{\mathsf{LF}}, k'_{\mathsf{lin}}, \cdot\right)$ for a freshly sampled $\mathsf{pk}_{\mathsf{LF}}$, k'_{lin}, instead of E_0. Since we are sending obfuscations $\left(\mathcal{P}, \mathcal{P}^{-1}, \mathcal{D}\right)$ of the three circuits and due to the three circuits accessing the samplers E_0, E_1 only as black boxes, it follows by Lemma 18 that the output of the previous hybrid and the current hybrid are $\left(f(\kappa), \frac{|\mathcal{X}|}{f(\kappa)}\right)$-indistinguishable, where $\mathcal{X}$ is the set of al possible values of y, which has size 2^r (recall that while the $y's$ are of length $d >> r$, the are a sparse set inside $\{0,1\}^d$ because are padding with zeros), and we also recall that κ, the cryptographic security parameter is some polynomial in the statistical security parameter λ. It follows in particular the success probability of the current process is $:= \epsilon_1 \geq \epsilon - \frac{2^r}{f(\kappa)} \geq \epsilon - \frac{\epsilon}{32} = \frac{31 \cdot \epsilon}{32}$.

- Hyb_2: Switching to a bounded number of cosets $(\mathbf{A}(y), \mathbf{b}(y))$, by using the lossy function.

The change from the previous hybrid to the current hybrid is that we are going to sample a lossy key $\mathsf{pk}^1_{\mathsf{LF}} \leftarrow \mathsf{LF.KeyGen}\left(1^d, 1, 1^w\right)$ and use it inside E_1 from the previous hybrid, instead of using an injective key $\mathsf{pk}^0_{\mathsf{LF}} \leftarrow \mathsf{LF.KeyGen}\left(1^d, 0, 1^w\right)$, which was used in the previous hybrid. Note that in this hybrid, there are at most 2^w cosets (that is, some different values of y will have the same coset), by the correctness of the lossy function scheme. By the security of the lossy function scheme, the output of this hybrid is $\left(f\left(w\right), \frac{1}{f(w)}\right)$-indistinguishable from the previous hybrid. It follows in particular the success probability of the current process is

$$:= \epsilon_2 \geq \epsilon_1 - \frac{1}{f\left(w\right)} \geq \frac{31 \cdot \epsilon}{32} - \frac{1}{f\left(w\right)} \geq \frac{30 \cdot \epsilon}{32} = \frac{15 \cdot \epsilon}{16} \quad .$$

- Hyb_3: Using obfuscated (instead of plain) circuits for membership checks inside D, by using the security of the obfuscator that obfuscates D.

Recall that in the previous hybrid, the circuit D executes as follows: $D(y, \mathbf{v})$ computes $\mathbf{A}(y)$ by access to $\mathsf{F}(k_{\mathsf{lin}}, \cdot)$, and then checks membership in the dual of $\mathsf{ColSpan}(\mathbf{A}(y)) := S_y$. In the current hybrid we make the following change to D: We sample an additional P-PRF key k_S at the beginning of the hybrid. Given $\mathbf{A}(y)$, we apply $\mathsf{F}(k_S, \cdot)$ to obtain pseudorandomness and generate an obfuscation $\mathsf{O}_{S_y^\perp} \leftarrow \mathsf{iO}\left(1^\kappa, S_y^\perp\right)$ of the circuit that checks membership inside $S_y^\perp$. The circuit D decides on the membership check of $\mathbf{v}$ using the obfuscated circuit $\mathsf{O}_{S_y^\perp}$ instead of the plain circuit $S_y^\perp$.

Note that by the correctness of the obfuscation scheme (specifically, the inner obfuscation scheme that was used to obfuscate the circuit $S_y^\perp$, for every y), we did not change the functionality of D. It follows by the security of the iO that obfuscates D, that the obfuscations between the two cases are $\left(f(\kappa), \frac{1}{f(\kappa)}\right)$-indistinguishable, and thus the same can be said of the outputs of the hybrids. It follows in particular the success probability of the current process is

$$:= \epsilon_3 \geq \epsilon_2 - \frac{1}{f(\kappa)} \geq \frac{15 \cdot \epsilon}{16} - \frac{\epsilon}{32} = \frac{29 \cdot \epsilon}{32} \quad .$$

- Hyb_4: Relaxing dual verification inside D to accept a larger subspace $T_y^\perp$ for every y, by using an puncturable puncturable PRF argument over subspace hiding.

We now consider two circuits in order to describe our current hybrid, both of which are used only inside the circuit D. The first circuit $E_S(k_S, \cdot)$ is the circuit that given an input $\mathbf{A}(y)$ applies $\mathsf{F}(k_S, \cdot)$ to obtain pseudorandomness and generate an obfuscation $\mathsf{O}_{S_y^\perp} \leftarrow \mathsf{iO}\left(1^\kappa, S_y^\perp\right)$. The second circuit $E_T(k_T, \cdot)$ is the circuit that given an input $\mathbf{A}(y)$ applies $\mathsf{F}(k_T, \cdot)$ to obtain pseudorandomness for two things: (1) to sample $T_y^\perp \subseteq \{0,1\}^k$ a superspace that contains $S_y^\perp$ and has s more dimensions (recall that S_y has $n-r$ dimensions, $S_y^\perp$ has $k-(n-r)$ dimensions and thus $T_y^\perp$ has $k-(n-r)+s$ dimensions), and (2) to generate an obfuscation $\mathsf{O}_{T_y^\perp} \leftarrow \mathsf{iO}\left(1^\kappa, T_y^\perp\right)$.

For concreteness, the way we use the randomness from $\mathsf{F}(k_T, \cdot)$ is by generating a pseudorandom full rank matrix $M_y \in \mathbb{Z}_2^{(n-r)\times(n-r)}$, multiplying by $\mathbf{A}(y) \in \mathbb{Z}_2^{k\times(n-r)}$ to get $\overline{\mathbf{A}}(y) := \mathbf{A}(y) \cdot M_y \in \mathbb{Z}_2^{k\times(n-r)}$. We then take the last $n-r-s$ columns of $\overline{\mathbf{A}}(y)$, denote this sub-matrix by $\overline{\mathbf{A}}^{(1)} \in \mathbb{Z}_2^{k\times(n-r-s)}$ and define $\mathsf{ColSpan}\left(\overline{\mathbf{A}}^{(1)}\right)^\perp := T_y^\perp$.

Let $\mathcal{X}$ the set of all possible cosets that arise from the scheme, which has size $\leq 2^w$ by the lossy function. We would first like to note the indistinguishability, per coset $i \in \mathcal{X}$, between $\mathsf{O}_{S_i^\perp}$ and $\mathsf{O}_{T_i^\perp}$, for a known S_i and uniformly random appropriate superspace T_i. Specifically, we look at the indistinguishability for truly random bits for sampling T and for sampling the obfuscation in either of the cases. For this, we would like to use Lemma 23. We would like to make sure the lemma's requirements are met so we note the dimensions with $'$. Note that in our case, k, $k-n+r$ and s, take the role of k', r' and s' respectively, and thus we get $\left(f(n-r-s), \frac{1}{f(n-r-s)}\right)$-indistinguishability.

Note that in the previous hybrid, the circuit E_S is used in D, and furthermore, the access of D to E_S is only as a black box. The change that we make to the current hybrid is that we are going to use $E_T(k_T, \cdot)$ for a freshly sampled P-PRF key k_T instead of E_S (which is also sampled for a freshly sampled P-PRF key k_S). Since we are sending an obfuscation $\mathcal{D}$ of D and due to D

accessing the samplers E_S, E_T only as black boxes, it follows by Lemma 18 that the output of the previous hybrid and the current hybrid are $\left(f\left(n-r-s\right), \frac{|\mathcal{X}|}{f(n-r-s)}\right)$-indistinguishable, where $\mathcal{X}$ is the set of all cosets, which has size $\leq 2^w$ by the lossy function. It follows in particular that the success probability of the current process is

$$:= \epsilon_4 \geq \epsilon_3 - \frac{2^w}{f\left(n-r-s\right)} \geq \frac{29\cdot\epsilon}{32} - \frac{\epsilon}{32} = \frac{28\cdot\epsilon}{32} \quad .$$

- Hyb_5: Asking for sum of collisions to be outside of T_y, by an obfuscated puncturable PRF argument over dual-subspace anti-concentration.

This hybrid is the same as the previous in terms of execution, but we change the definition of a successful execution, that is, we change the predicate computed on the output of the process. We still ask that $(y_0 = y_1 := y)$, but instead of only asking the second requirement to be $(x_0 \neq x_1)$, we ask for a stronger condition: $(\mathbf{u}_0 - \mathbf{u}_1) \in (S_y \setminus T_y)$. Note that we are not going to need to be able to efficiently check for the success of the condition, but we'll prove that it happens with a good probability nonetheless.

Let ϵ_5 be the success probability of the current hybrid and note that $\mathcal{A}$ finds collisions with probability ϵ_4 in the previous hybrid Hyb_4 (and since this hybrid is no different, the same goes for the current hybrid). Let $\mathcal{X} \subseteq \{0,1\}^m$ the image of the lossy function $\mathsf{LF.F}\left(\mathsf{pk}_{\mathsf{LF}}, \cdot\right)$ which we use to map our images y to cosets $(\mathbf{A}(y), \mathbf{b}(y))$, that is, there are $|\mathcal{X}|$ cosets and by the lossyness we know that $|\mathcal{X}| \leq 2^w$. For every value $\mathbf{x} \in \mathcal{X}$ denote by $\epsilon_4^{(\mathbf{x})}$ the probability to find a collision on value $\mathbf{x}$, or formally, that $y_0 = y_1 := y$, $x_0 \neq x_1$ and $\mathbf{x} = \mathsf{LF.F}\left(\mathsf{pk}_{\mathsf{LF}}, y\right)$. We deduce $\sum_{\mathbf{x}\in\mathcal{X}} \epsilon_4^{(\mathbf{x})} = \epsilon_4$. Let L be a subset of $\mathcal{X}$ such that $\epsilon_4^{(\mathbf{x})} \geq \frac{\epsilon_4}{2\cdot|\mathcal{X}|}$ and note that $\sum_{\mathbf{x}\in L} \epsilon_4^{(\mathbf{x})} \geq \frac{\epsilon_4}{2}$. We further define $\epsilon_5^{(\mathbf{x})}$ as the probability to find a strong (as in the notion of ϵ_5) collision on value $\mathbf{x}$, or formally, that $y_0 = y_1 := y$, $(\mathbf{u}_0 - \mathbf{u}_1) \in (S_y \setminus T_y)$ and $\mathbf{x} = \mathsf{LF.F}\left(\mathsf{pk}_{\mathsf{LF}}, y\right)$. Note that S_y, T_y are really functions of $\mathbf{x}$ rather than of y, so $(\mathbf{u}_0 - \mathbf{u}_1) \in (S_{\mathbf{x}} \setminus T_{\mathbf{x}})$ and also observe that $\sum_{\mathbf{x}\in\mathcal{X}} \epsilon_5^{(\mathbf{x})} = \epsilon_5$.

We would now like to use Lemma 26, so we make sure that we satisfy its requirements. Let any $\mathbf{x} \in L$, we know that by definition $\epsilon_4^{(\mathbf{x})} \geq \frac{\epsilon_4}{2\cdot|\mathcal{X}|}$ and also recall that $\epsilon_4 \geq \frac{28\cdot\epsilon}{32}$, $|\mathcal{X}| \leq 2^w$ and thus

$$\epsilon_4^{(\mathbf{x})} \geq \frac{\epsilon_4}{2\cdot|\mathcal{X}|} \geq \frac{28\cdot\epsilon}{64}\cdot\frac{1}{2^w} \geq \Omega\left(\frac{\epsilon}{2^w}\right) \quad .$$

Let $s' := s - (n-r-s)$ and for any $\mathbf{x} \in L$ let $\ell_{\mathbf{x}} := \frac{k^2}{\epsilon_4^{(\mathbf{x})}} \leq O\left(\frac{k^2\cdot 2^w}{\epsilon}\right)$. Note that by our Lemma 54 statement's assumptions, we have (1) $\frac{(n-r-s)\cdot\frac{1}{\epsilon_4^{(\mathbf{x})}}}{2^{s'}} \leq o(1)$ and (2) $\frac{\ell_{\mathbf{x}}\cdot\left(k^3+\mathsf{poly}(n-r-s)+T_{\mathcal{A}}\right)}{f(n-r-s)} \leq o(1)$. Since this satisfies Lemma 26, it follows that for every $\mathbf{x} \in L$ we have $\epsilon_5^{(\mathbf{x})} \geq \frac{\epsilon_4^{(\mathbf{x})}}{16\cdot k^2} - \frac{1}{f(\kappa)}$, because for each $\mathbf{x} \in \mathcal{X}$, in order to use Lemma 26, we need the randomness for the experiment to be genuinely random, which will necessitate us to invoke the security of the iO and puncturable PRF, which incurs the loss of $\frac{1}{f(\kappa)}$. It follows that

$$\epsilon_5 = \sum_{\mathbf{x}\in\mathcal{X}} \epsilon_5^{(\mathbf{x})} \geq \sum_{\mathbf{x}\in L} \epsilon_5^{(\mathbf{x})} \geq \sum_{\mathbf{x}\in L} \frac{\epsilon_4^{(\mathbf{x})}}{16\cdot k^2} - \frac{|\mathcal{X}|}{f(\kappa)} \geq \frac{\left(\frac{\epsilon_4}{2}\right)}{16\cdot k^2} - \frac{|\mathcal{X}|}{f(\kappa)} \geq \frac{\left(\frac{28\cdot\epsilon}{64}\right)}{16\cdot k^2} - \frac{|\mathcal{X}|}{f(\kappa)} \geq \frac{\epsilon}{64\cdot k^2} \quad .$$

- Hyb_6: For every y, de-randomizing T_y and defining it as the column span of $\mathbf{A}(y)^{(1)} \in \mathbb{Z}_2^{k\times(n-r-s)}$, the last $n-r-s$ columns of the matrix $\mathbf{A}(y)$, by using permutable PRPs, security of iO and an obfuscated puncturable PRF.

This hybrid is the same as the previous, with the following change. Recall how we compute dual membership check inside the circuit D: Given y, we compute $\mathbf{x} = \mathsf{LF.F}(\mathsf{pk}_{\mathsf{LF}}, y)$ we take the P-PRF key k_{in} and compute $\mathbf{A}(\mathbf{x}) = \mathbf{A}(y)$, then use the additional P-PRF key k_T to generate the random superspace $T_y^{\perp}$ of $S_y^{\perp}$ and additional randomness, and use both to generate an obfuscation $\mathsf{O}_{T_y^{\perp}} \leftarrow \mathsf{iO}\left(1^{\lambda}, T_y^{\perp}\right)$. Also recall that the subspace $T_y^{\perp}$ is generated as follows: Generate the matrix M_y using the P-PRF, then multiply it by $\mathbf{A}(y)$ to get $\overline{\mathbf{A}}(y)$, and then take the last $n-r-s$ columns of it as a basis for T_y, and $T_y^{\perp}$ is defined as the dual of that space. In the current hybrid we will not sample M_y and simply define $\overline{\mathbf{A}}_y := \mathbf{A}_y$. This means that T_y is defined to be the column span of $\mathbf{A}(y)^{(1)}$. We will also not obfuscate the membership check for $T_y^{\perp}$ inside the circuit D, and check membership by using a basis of $T_y^{\perp}$ in the plain. We next define hybrid experiments (which go in the direction from the previous hybrid Hyb_5 to the current hybrid Hyb_6) and explain why each consecutive pair are appropriately indistinguishable.

- **Using the security of the permutable PRP.** Assume we sample all components of our scheme, excluding the key k_{in} for the initial permutation Π on $\{0,1\}^n$, and define the following permutation Γ on $\{0,1\}^n$. Denote by h the first r bits of the input and by j the last $n-r$ bits of the input to the permutation Γ. Recall that in the circuits P, P^{-1}, D, the value y, the coset $\mathbf{A}(y), \mathbf{b}(y)$, the superspace $T_y^{\perp}$ and the associated matrix $M_y \in \mathbb{Z}_2^{(n-r)\times(n-r)}$, are all computed as a function of r bits, which in the construction take the role of $H(x)$. In fact, all of the above variables can be written as a function of $h \in \{0,1\}^r$ instead of as a function of y. The permutation Γ takes h and computes M_h (the matrix for computing T_h from the matrix $\mathbf{A}(h)$), interprets j as a vector in $\mathbb{Z}_2^{n-r}$, and applies M_h to j. Note that this change means we are taking the puncturable PRF key k_T which is used to sample $T_y^{\perp}$ (and formally, is used to sample the pseudorandomness for generating the matrix M_y and for obfuscating membership check for $T_y^{\perp}$), which previously only existed inside the circuit D, and putting it also inside the circuits P, P^{-1}, because this key is needed in order to compute the permutation Γ.

  Recall the examples of decomposable permutations in Section 5. For every value $h \in \{0,1\}^r$ observe that multiplication by M_h is a permutation (and moreover an affine permutation, which is decomposable efficiently). Then, Γ is a controlled permutation as described in the examples of decomposable permutations in Section 5, which is controlled on decomposable permutations. Overall, we deduce that Γ is a $\left(2^{\mathsf{poly}(\lambda)}, \mathsf{poly}(\lambda)\right)$-decomposable permutation. We use the permutable PRP Π, and switch to a setting where we use the key k_{in}^{Γ} that applies Γ to the output of Π (and Γ^{-1} to the input of Π), instead of just applying Π and its inverse. By the security of the permutable PRP, this change is $\left(f(\kappa), \frac{1}{f(\kappa)}\right)$-indistinguishable.

- **Using the security of outside iO.** For every $h \in \mathbb{Z}_2^r$, we make the following change to P, P_1. Instead of composing the permutation Γ to the output of Π, it applies Π as it is. However, when generating the matrix $\mathbf{A}(y)$, it multiplies by M_y to get $\overline{\mathbf{A}}(y) := \mathbf{A}(y) \cdot M_y$. An additional change we will make not to P, P^{-1} but to the circuit D, is that we will not obfuscate the membership check circuit for $T_y^{\perp}$ and simply use its available basis. One can observe that we did not change the functionality of P, P^{-1}, D in any of the above changes and

thus by the security of the indistinguishability obfuscation that obfuscates $\mathcal{P} \leftarrow \mathsf{iO}\left(1^\kappa, P\right)$, $\mathcal{P}^{-1} \leftarrow \mathsf{iO}\left(1^\kappa, P^{-1}\right)$, $\mathcal{D} \leftarrow \mathsf{iO}\left(1^\kappa, D\right)$, this change is $\left(f(\kappa), \frac{1}{f(\kappa)}\right)$-indistinguishable.

- **Using an obfuscated puncturable PRF argument over the choice of the matrix $\mathbf{A}(y)$, for every y.** Note that after we did the last step, our process for generating the cosets, in all tree circuits P, P^{-1}, D is the following: Given y we compute $\mathbf{x}$ with the lossy function, and then apply two different puncturable PRFs (that is, with i.i.d keys, k_{lin} and k_T) to obtain $(\mathbf{A}(\mathbf{x}), \mathbf{b}(\mathbf{x}))$ and the matrix $M_{\mathbf{x}} \in \mathbb{Z}_2^{(n-r)\times(n-r)}$. We then multiply to get the matrix that we are actually using, i.e., $\overline{\mathbf{A}}(\mathbf{x}) := \mathbf{A}(\mathbf{x}) \cdot M_{\mathbf{x}}$. The change we next make is to have one fresh puncturable PRF key k'_{lin} and use it to generate the coset $(\mathbf{A}(\mathbf{x}), \mathbf{b}(\mathbf{x}))$, without further generating the matrix $M_{\mathbf{x}}$.

  By a standard argument using an obfuscated puncturable PRF like we used numerous times in this proof (i.e., we use lemma 18 and the fact that when using real randomness, the two ways to sample $\mathbf{A}(y)$ are statistically equivalent), we get that this change is $\left(f(\kappa), 2^w \cdot \frac{1}{f(\kappa)}\right)$-indistinguishable.

Observe that after the last change we are exactly in the setting of Hyb_6, and we started in Hyb_5. It follows in particular that the success probability of the current process is

$$:= \epsilon_6 \geq \epsilon_5 - \frac{2^w}{f\left(\kappa\right)} \geq \frac{\epsilon}{64 \cdot k^2} - \frac{\epsilon}{128 \cdot k^2} = \frac{\epsilon}{128 \cdot k^2} \ .$$

- Hyb_7: Going back to using 2^r cosets rather than $\leq 2^w$, by moving from lossy mode to injective mode in the lossy function.

We make the exact same change we made between Hyb_1 to Hyb_2, but in the opposite direction. That is we sample an injective key $\mathsf{pk}^0_{\mathsf{LF}} \leftarrow \mathsf{LF.KeyGen}\left(1^d, 0, 1^w\right)$ and use it instead of the previous lossy key $\mathsf{pk}^1_{\mathsf{LF}} \leftarrow \mathsf{LF.KeyGen}\left(1^d, 1, 1^w\right)$, which is used in the previous hybrid. By the exact same argument (which relies on the security of the lossy function), the output of this hybrid is $\left(f(w), \frac{1}{f(w)}\right)$-indistinguishable. It follows in particular the success probability of the current process is

$$:= \epsilon_7 \geq \epsilon_6 - \frac{1}{f\left(w\right)} \geq \frac{\epsilon}{128 \cdot k^2} - \frac{\epsilon}{256 \cdot k^2} = \frac{\epsilon}{256 \cdot k^2} \ .$$

- Hyb_8: Stop using the lossy function, by an obfuscated puncturable PRF argument.

We make the exact same change we made between Hyb_0 to Hyb_1, but in the opposite direction. That is, we drop the lossy function $\mathsf{LF.F}$ altogether and apply the PRF to y directly and not to $\mathbf{x}$, the output of the lossy function on input y. By the exact same argument (which relies on Lemma 18), the output of this hybrid is $\left(f(\kappa), \frac{|\mathcal{X}|}{f(\kappa)}\right)$-indistinguishable, where $\mathcal{X}$ is the set of al possible values of y, which has size 2^r (recall that while the $y's$ are of length $d >> r$, the are a sparse set inside $\{0,1\}^d$ because are padding with zeros). It follows in particular the success probability of the current process is

$$:= \epsilon_8 \geq \epsilon_7 - \frac{2^r}{f\left(\kappa\right)} \geq \frac{\epsilon}{256 \cdot k^2} - \frac{\epsilon}{512 \cdot k^2} = \frac{\epsilon}{512 \cdot k^2} \ .$$

To conclude, note that the generated obfuscations in the final hybrid Hyb_8 form exactly the distribution $\left(\mathcal{P}, \mathcal{P}^{-1}, \mathcal{D}'\right) \leftarrow \widetilde{\mathsf{Gen}}(1^\lambda, n, r, k, s)$. This finishes our proof. □

6.2 Simulating the Dual

Our next step is to show that an adversary which has access to the dual-free setting can simulate the CRS for an adversary in the restricted setting, where the dual verification check is bloated.

Lemma 55. *Let $\lambda \in \mathbb{N}$ and assume there is a quantum algorithm $\mathcal{A}$ running in time $T_{\mathcal{A}}$ such that,*

$$\Pr\left[\begin{array}{l} y_0 = y_1 := y, \\ (\mathbf{u}_0 - \mathbf{u}_1) \notin \mathsf{ColSpan}\left(\mathbf{A}^{(1)}(y)\right) \end{array} : \begin{array}{rl} \left(\mathcal{P}, \mathcal{P}^{-1}, \mathcal{D}'\right) & \leftarrow \widetilde{\mathsf{Gen}}(1^\lambda, n, r, k, s) \\ (x_0, x_1) & \leftarrow \mathcal{A}\left(\mathcal{P}, \mathcal{P}^{-1}, \mathcal{D}'\right) \\ (y_b, \mathbf{u}_b) & \leftarrow \mathcal{P}(x_b) \end{array}\right] \geq \epsilon \ .$$

Assume that the primitives used in Construction 52 are $\left(f(\cdot), \frac{1}{f(\cdot)}\right)$-secure, and assume $\frac{\frac{2^r}{\epsilon}}{f(\kappa)} \leq o(1)$. Then, there is a quantum algorithm $\mathcal{B}$ running in time $T_{\mathcal{A}} + \mathsf{poly}(\lambda)$ such that

$$\Pr\left[(\overline{y}_0 = \overline{y}_1 := \overline{y}) \wedge (\overline{x}_0 \neq \overline{x}_1) \ : \begin{array}{rl} \left(\overline{\mathcal{P}}, \overline{\mathcal{P}}^{-1}, \overline{\mathcal{D}}\right) & \leftarrow \widetilde{\mathsf{Gen}}\left(1^\lambda, r+s, r, k-(n-r-s), 0\right) \\ (\overline{x}_0, \overline{x}_1) & \leftarrow \mathcal{B}\left(\overline{\mathcal{P}}, \overline{\mathcal{P}}^{-1}\right) \\ (\overline{y}_b, \overline{\mathbf{u}}_b) & \leftarrow \overline{\mathcal{P}}(x_b) \end{array}\right] \geq \frac{\epsilon}{2} \ .$$

Proof. We first describe the actions of the algorithm $\mathcal{B}$ (which will use the code of $\mathcal{A}$ as part of its machinery) and then argue why it breaks collision resistance with the appropriate probability. Given $\overline{\mathcal{P}}, \overline{\mathcal{P}}^{-1}$ which comes from $\left(\overline{\mathcal{P}}, \overline{\mathcal{P}}^{-1}, \overline{\mathcal{D}}\right) \leftarrow \widetilde{\mathsf{Gen}}\left(1^\lambda, r+s, r, k-(n-r-s), 0\right)$, the algorithm $\mathcal{B}$ does the following:

- Sample a P-PRF key $k_{\mathbf{C}}$ that outputs some sufficient (polynomial) amount of random bits on an d-bit input, and sample a permutable PRP key k_Γ for a PRP on domain $\{0,1\}^n$. Define the following circuits.
- $\left(\ y \in \mathbb{Z}_2^d,\ \mathbf{u} \in \mathbb{Z}_2^k\ \right) \leftarrow P\left(x \in \mathbb{Z}_2^n\right)$:
  - $\left(\ \overline{x} \in \mathbb{Z}_2^{r+s},\ \widetilde{x} \in \mathbb{Z}_2^{n-r-s}\ \right) \leftarrow \Pi\left(k_\Gamma, x\right)$.
  - $\left(\ y \in \mathbb{Z}_2^d,\ \overline{\mathbf{u}} \in \mathbb{Z}_2^{k-(n-r-s)}\ \right) \leftarrow \overline{\mathcal{P}}(\overline{x})$.
  - $\left(\ \mathbf{C}(y) \in \mathbb{Z}_2^{k \times k},\ \mathbf{d}(y) \in \mathbb{Z}_2^{n-r-s}\right) \leftarrow \mathsf{F}\left(k_{\mathbf{C}}, y\right)$.
  - $\mathbf{u} \leftarrow \mathbf{C}(y) \cdot \begin{pmatrix} \overline{\mathbf{u}} \\ \widetilde{x} + \mathbf{d}(y) \end{pmatrix}$.
- $\left(\ x \in \mathbb{Z}_2^n\ \right) \leftarrow P^{-1}\left(\ y \in \mathbb{Z}_2^d,\ \mathbf{u} \in \mathbb{Z}_2^k\ \right)$:
  - $\left(\ \mathbf{C}(y) \in \mathbb{Z}_2^{k \times k},\ \mathbf{d}(y) \in \mathbb{Z}_2^{n-r-s}\right) \leftarrow \mathsf{F}\left(k_{\mathbf{C}}, y\right)$.
  - $\begin{pmatrix} \overline{\mathbf{u}} \\ \widetilde{x} \end{pmatrix} \leftarrow \mathbf{C}(y)^{-1} \cdot \mathbf{u} - \begin{pmatrix} 0^{k-(n-r-s)} \\ \mathbf{d}(y) \end{pmatrix}$.
  - $\left(\ \overline{x} \in \mathbb{Z}_2^{r+s}\ \right) \leftarrow \overline{\mathcal{P}}^{-1}\left(y, \overline{\mathbf{u}}\right)$.
  - $x \leftarrow \Pi^{-1}\left(k_\Gamma, (\overline{x}, \widetilde{x})\right)$.
- $D'\left(\ y \in \mathbb{Z}_2^d,\ \mathbf{v} \in \mathbb{Z}_2^k\ \right) \in \{0,1\}$:

- $\left(\mathbf{C}(y) \in \mathbb{Z}_2^{k \times k},\ \mathbf{d}(y) \in \mathbb{Z}_2^{n-r-s} \right) \leftarrow \mathsf{F}\left(k_{\mathbf{C}}, y\right)$.
- $\mathbf{A}^{(1)}(y) :=$ last $n-r-s$ columns of $\mathbf{C}(y)$.
- Output 1 iff $\mathbf{v}^T \cdot \mathbf{A}^{(1)}(y) = \mathbf{0}^{n-r-s}$.

- Use indistinguishability obfuscation in order to generate the input for $\mathcal{A}$: $\mathcal{P} \leftarrow \mathsf{iO}\left(1^\kappa, P\right)$, $\mathcal{P}^{-1} \leftarrow \mathsf{iO}\left(1^\kappa, P^{-1}\right)$, $\mathcal{D}' \leftarrow \mathsf{iO}\left(1^\kappa, D'\right)$.

The remainder of the reduction is simple: $\mathcal{B}$ executes $(x_0, x_1) \leftarrow \mathcal{A}\left(\mathcal{P}, \mathcal{P}^{-1}, \mathcal{D}'\right)$ and then $(\overline{x}_b, \widetilde{x}_b) \leftarrow \Pi\left(k_\Gamma, x_b\right)$ and outputs $(\overline{x}_0, \overline{x}_1)$. Assume that the output of $\mathcal{A}$ satisfies $y_0 = y_1 := y$ and also $(\mathbf{u}_0 - \mathbf{u}_1) \notin \mathsf{ColSpan}\left(\mathbf{A}(y)^{(1)}\right)$, and recall that $\mathbf{A}(y)^{(1)} \in \mathbb{Z}_2^{k \times (n-r-s)}$ are the last $n-r-s$ columns of the matrix $\mathbf{A}(y) \in \mathbb{Z}_2^{k \times (n-r)}$, which is generated by the reduction. We explain why it is necessarily the case that $\overline{x}_0 \neq \overline{x}_1$.

First note that due to how we defined the reduction, $\mathbf{A}(y) := \mathbf{C}(y) \cdot \begin{pmatrix} \overline{\mathbf{A}}(y) & \\ & \mathbf{I}_{n-r-s} \end{pmatrix}$, where $\overline{\mathbf{A}}(y) \in \mathbb{Z}_2^{(k-(n-r-s)) \times s}$ is the matrix arising from $\overline{\mathcal{P}}, \overline{\mathcal{P}}^{-1}$ and $\mathbf{I}_{n-r-s} \in \mathbb{Z}_2^{(n-r-s) \times (n-r-s)}$ is the identity matrix of dimension $n-r-s$. Also note that because $\mathbf{C}(y)$, $\overline{\mathbf{A}}(y)$ are full rank then $\mathbf{A}(y)$ is full rank. Now, since $(\mathbf{u}_0 - \mathbf{u}_1) \notin \mathsf{ColSpan}\left(\mathbf{A}(y)^{(1)}\right)$ and since $\mathbf{A}(y)^{(1)}$ are the last $n-r-s$ columns of $\mathbf{A}(y)$, it follows that if we consider the coordinates vector $\mathbf{x} \in \mathbb{Z}_2^{n-r}$ of $(\mathbf{u}_0 - \mathbf{u}_1)$ with respect to $\mathbf{A}(y)$, the first s elements are not 0^s. By linearity of matrix multiplication it follows that if we look at each of the two coordinates vectors $\mathbf{x}_0$, $\mathbf{x}_1$ (each has $n-r$ bits) for $\mathbf{u}_0$, $\mathbf{u}_1$, respectively, somewhere in the first s bits, they differ. Now, recall how we obtain the first s bits of $\mathbf{x}_b$ – this is exactly by applying $\overline{\Pi}$ (the permutation on $\{0,1\}^{r+s}$ arising from $\overline{\mathcal{P}}, \overline{\mathcal{P}}^{-1}$) to $\overline{x}_b$ and taking the last s bits of the output. Since these bits differ in the output of the permutation, then the preimages have to differ, i.e., $\overline{x}_0 \neq \overline{x}_1$.

Define $\epsilon_{\mathcal{B}}$ as the probability that the output of $\mathcal{A}$ indeed satisfies $y_0 = y_1 := y$ and also $(\mathbf{u}_0 - \mathbf{u}_1) \notin \mathsf{ColSpan}\left(\mathbf{A}^{(1)}(y)\right)$, and it remains to give a lower bound for the probability $\epsilon_{\mathcal{B}}$. We do this by a sequence of hybrids, eventually showing that the view which $\mathcal{B}$ simulates to $\mathcal{A}$ is computationally indistinguishable from a sample from $\widetilde{\mathsf{Gen}}\left(1^\lambda, n, r, k, s\right)$. More precisely, each hybrid describes a process, it has an output, and a success predicate on the output.

- Hyb_0: The above distribution $\left(\mathcal{P}, \mathcal{P}^{-1}, \mathcal{D}'\right) \leftarrow \mathcal{B}\left(\overline{\mathcal{P}}, \overline{\mathcal{P}}^{-1}\right)$, simulated to the algorithm $\mathcal{A}$.

The first distribution is defined in the reduction above. The output of the process is the output (x_0, x_1) of $\mathcal{A}$. The process execution is considered as successful if $y_0 = y_1 := y$ and $(\mathbf{u}_0 - \mathbf{u}_1) \notin \mathsf{ColSpan}\left(\mathbf{A}(y)^{(1)}\right)$.

- Hyb_1: Not applying the inner permutation $\overline{\Pi}_{\mathsf{in}}$ (which comes from the circuits $\overline{\mathcal{P}}$, $\overline{\mathcal{P}}^{-1}$), by using the security of an obfuscated permutable PRP.

Let $\overline{\Pi}_{\mathsf{in}}$ the (first) permutable PRP that's inside $\overline{\mathcal{P}}$ (which is the obfuscation of the circuit $\overline{P}$). In the previous hybrid we apply the n-bit permutable PRP $\Pi\left(k_\Gamma, \cdot\right)$ to the input $x \in \mathbb{Z}_2^n$ and then proceed to apply the inner permutation $\overline{\Pi}_{\mathsf{in}}\left(\overline{k}_{\mathsf{in}}, \cdot\right)$ to the first (i.e. leftmost) $r+s$ output bits of the first permutation $\Pi\left(k_\Gamma, \cdot\right)$. The change we make to the current hybrid is that we simply apply only $\Pi\left(k_\Gamma, \cdot\right)$.

Recall two details: (1) By Theorem 39, the inner permutable PRP $\overline{\Pi}_{\mathsf{in}}\left(\overline{k}_{\mathsf{in}},\cdot\right)$ is in and of itself $\left(2^{\mathsf{poly}(\lambda)},\mathsf{poly}(\lambda)\right)$-decomposable, and (2) the circuits P, P^{-1} which apply the permutations are both obfuscated by iO to be generate the obfuscations $\mathcal{P}$, $\mathcal{P}^{-1}$. We can treat it as a fixed permutation that acts on the output of the permutation $\Pi\left(k_{\Gamma},\cdot\right)$ and thus it follows by Lemma 40 that the current and previous hybrids are computationally indistinguishable, with indistinguishability $\frac{1}{f(\kappa)}$.

- Hyb_2: For every $y\in\mathbb{Z}_2^d$, taking $\mathbf{A}(y)$ to be the direct output of the PRF F, by using an obfuscated punctured PRF argument.

In order to describe the change between the current and previous hybrid we first recall the structure of the circuits from the previous hybrid: In the previous hybrid, for every $y\in\mathbb{Z}_2^r$ we defined $\mathbf{A}(y):=\mathbf{C}(y)\cdot\begin{pmatrix}\overline{\mathbf{A}}(y)&\\&\mathbf{I}_{n-r-s}\end{pmatrix}$, where $\mathbf{C}(y)\in\mathbb{Z}_2^{k\times k}$ comes from the output $\mathsf{F}\left(k_{\mathbf{C}},y\right)$ and $\overline{\mathbf{A}}(y)\in\mathbb{Z}_2^{(k-(n-r-s))\times s}$ is the output of the inner PRF $\overline{\mathsf{F}}\left(\overline{k}_{\mathsf{lin}}\right)$ (which in turn comes from the inside of $\left(\overline{\mathcal{P}},\overline{\mathcal{P}}^{-1}\right)$). In the current hybrid we are going to ignore the PRFs $\mathsf{F}\left(k_{\mathbf{C}},y\right)$ and $\overline{\mathsf{F}}\left(\overline{k}_{\mathsf{lin}}\right)$ and their generated values $\mathbf{C}(y)$, $\mathbf{d}(y)$ and $\overline{\mathbf{A}}(y)$ and instead, sample a fresh key $k_{\mathbf{A}}$, and on query y generate $\mathbf{A}(y)\leftarrow\mathsf{F}\left(k_{\mathbf{A}},y\right)$, for $\mathbf{A}(y)\in\mathbb{Z}_2^{k\times(n-r)}$.

First, note that the following two ways to sample $\mathbf{A}(y)$, are statistically equivalent for every y: (1) The matrix $\mathbf{A}(y)$ is generated by sampling a random full-rank matrix $\mathbf{C}(y)\in\mathbb{Z}_2^{k\times k}$ and letting $\mathbf{A}(y)$ be $\mathbf{C}(y)\cdot\begin{pmatrix}\overline{\mathbf{A}}(y)&\\&\mathbf{I}_{n-r-s}\end{pmatrix}$. (2) For every $y\in\mathbb{Z}_2^r$ just sample a full-rank matrix $\mathbf{A}(y)\in\mathbb{Z}_2^{k\times(n-r)}$. This means that when truly random bits are used for generating $\mathbf{A}(y)$ in the two cases, the distributions are statistically equivalent.

To see why the two distributions are computationally indistinguishable, a different description of the previous hybrid can be given as follows: We can consider a sampler E_0 that for every $y\in\{0,1\}^d$ samples $\mathbf{A}(y)$ according to the first algorithm, and another sampler E_1 that samples $\mathbf{A}(y)$ according to the second algorithm, and we know that for every y (and recall there are 2^r actual values of y which can appear as the output, and not 2^d) the outputs of E_0 and E_1 are statistically indistinguishable.

Since there are 2^r valid values for y, by Lemma 18, the current hybrid is computationally indistinguishable from the previous, with indistinguishability $\frac{2^r}{f(\kappa)}$.

- Hyb_3: Discarding the inner obfuscations $\left(\overline{\mathcal{P}},\overline{\mathcal{P}}^{-1}\right)$ completely, by using the security of the outer obfuscator.

The change between the current hybrid and the previous is that in the current hybrid we generate the circuits P,P^{-1},D' without using $\left(\overline{\mathcal{P}},\overline{\mathcal{P}}^{-1}\right)$ at all. Note that this is possible, since in the previous hybrid, we moved to a circuit that did not use access to the circuits $\left(\overline{\mathcal{P}},\overline{\mathcal{P}}^{-1}\right)$ any longer during the execution of any of the three circuits P,P^{-1},D', except from using the second permutation $\overline{\Pi}_{\mathsf{out}}$, which acts on $\{0,1\}^d$ and does not need to act from inside the inner obfuscations $\left(\overline{\mathcal{P}},\overline{\mathcal{P}}^{-1}\right)$ any more. This means that we can technically move the application of the inner permutation $\overline{\Pi}_{\mathsf{out}}$ "outside of the inner circuits $\left(\overline{\mathcal{P}},\overline{\mathcal{P}}^{-1}\right)$" and the functionality of the circuits P,P^{-1},D' did not

change between the current and the previous hybrids, and thus, by the security of the indistinguishability obfuscator that obfuscates the three circuits, the current hybrid is computationally indistinguishable from the previous one, with indistinguishability of $\frac{1}{f(\kappa)}$.

Finalizing the reduction. Finally, observe that the distribution generated in the above Hyb_3 is exactly a sample from $\widetilde{\mathsf{Gen}}\left(1^\lambda, n, r, k, s\right)$. Also observe that the outputs of Hyb_0 and Hyb_3 are $O\left(\frac{2^r}{f(\kappa)}\right)$-computationally indistinguishable. Recall that by the lemma's assumptions, with probability ϵ, on a sample from $\widetilde{\mathsf{Gen}}\left(1^\lambda, n, r, k, s\right)$, the algorithm $\mathcal{A}$ outputs a pair (x_0, x_1) of n-bit strings such that $y_0 = y_1 := y$ and also $(\mathbf{u}_0 - \mathbf{u}_1) \notin \mathsf{ColSpan}\left(\mathbf{A}^{(1)}(y)\right)$. It follows that the probability for the same event when the input to $\mathcal{A}$ is generated by Hyb_0, is at least $\epsilon - O\left(\frac{2^r}{f(\kappa)}\right) \geq \frac{\epsilon}{2}$, which finishes our proof. □

6.3 Hardness of the Dual-free Case from LWE and iO

Here, we explain how to prove the collision resistance of the dual-free case (where the adversary sees $\mathcal{P}, \mathcal{P}^{-1}$, but not $\mathcal{D}$). This will follow the blueprint used in Section 4.3 to reduce this case to a standard collision-resistance problem. However, a few technical challenges arise. Some of these are due to needing certain steps to be efficient, where they are naively inefficient in Section 4.3. Another issue is that the underlying 2-to-1 function we use based on LWE [BCM+18] is not uniformly 2-to-1 but rather has some points that have no collisions. By careful arguments, we are nevertheless able to resolve these issues and prove security.

LWE-based approximate 2-to-1 functions. Here, we recall an *approximate* 2-to-1 function based on LWE which is a simplified version of the *noisy claw-free trapdoor function* developed in [BCM+18]. Let $u, v, \sigma, B, \overline{B}, q$ be parameters with the relationships described in Equation 1.

$$\begin{aligned} \sigma &= u^{\Omega(1)} & \overline{B} &= \sigma \times u^{\Omega(1)} \\ B &\geq \overline{B} \times u^{\omega(1)} & q &\geq B \times u^{\Omega(1)} \\ v &\geq \Omega(u \log q) \end{aligned} \tag{1}$$

The keys for the hash function have the form $k = (\mathbf{B}, \mathbf{c})$, where $\mathbf{B} \leftarrow \mathbb{Z}_q^{v \times u}$ and $\mathbf{c} \leftarrow \mathbf{B} \cdot \mathbf{s} + \mathbf{e} \bmod q$ where $\mathbf{s} \leftarrow \mathbb{Z}_q^u$ and the entries of $\mathbf{e} \in \mathbb{Z}_q^m$ are sampled from discrete Gaussians of width σ, which are guaranteed (whp) to have entries in $(-\overline{B}, \overline{B}]$. Let $\mathbf{B}' = (\mathbf{B} \,|\mathbf{I}_v)$.

The domain for $Q(k, \cdot)$ is $\{0,1\} \times \mathbb{Z}_q^u \times (-B, B]^m$. We then define $Q((\mathbf{B}, \mathbf{c}), (b, \mathbf{t}, \mathbf{f})) = b\mathbf{c} + \mathbf{B} \cdot \mathbf{t} + \mathbf{f} \bmod q$. We can equivalently write this as $Q((\mathbf{B}', \mathbf{c}), (b, \mathbf{t}')) = b\mathbf{c} + \mathbf{B}' \cdot \mathbf{t}' \bmod q$, where $\mathbf{t}' = (\mathbf{t}, \mathbf{f})$. By choosing B, q to be powers of 2, we can map the domain and range to bit-strings.

Let $\mathbf{s}' = (\mathbf{s}, \mathbf{e})$. Observe that $Q(k, \cdot)$ is *almost* 2-to-1, as any tuple $(b, \mathbf{t}')$ will collide with $(0, \mathbf{t}' + (-1)^b \mathbf{s}') \bmod q$. Moreover, by our choice of $v \geq \Omega(u \log q)$, with overwhelming probability over the choice of $\mathbf{B}'$, these will be the *only* type of collision. The only issue is that the colliding input may lie outside the domain, namely if $(-1)^b \mathbf{e} + \mathbf{f} \notin (-B, B]$. Fortunately since B is very large relative to $\mathbf{e}$, the vast majority of the domain is 2-to-1. However, for various iO techniques, we need to *exactly* match the functionality, so we will ultimately need to figure out how to deal with these bad points.

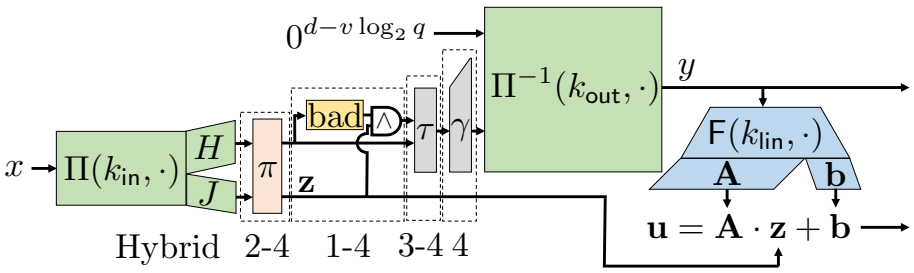

Figure 8: The program P in Hybrids 1-4. The output of $\Pi(k_{\mathsf{in}},\cdot)$ is divided into slices $(b,\mathbf{t},\mathbf{f})$ where $J(x)$ contains all of the b's, and $H(x)$ contains all of the $\mathbf{t},\mathbf{f}$. π maps each slice $(b,\mathbf{t},\mathbf{f}) \mapsto (b,\mathbf{t}',\mathbf{f}') = (b,\mathbf{t}+b\mathbf{s} \bmod q, +\mathbf{be} \bmod (-\mathbf{B},\mathbf{B}]$. bad determines if $\mathbf{f}'-\mathbf{e} \notin (-B,B]^n$; let b' be the b in that case or 0 otherwise. τ maps each slice $(\mathbf{t}',\mathbf{f}',b') \mapsto (\mathbf{t}',\mathbf{f}'') = (\mathbf{t}',[\ \mathbf{f}'-b'\mathbf{e} \bmod (-B,B]\ ]+b'\mathbf{e}$, where the final $+b'\mathbf{e}$ is not reduced. Finally, γ maps each slice $(\mathbf{t}',\mathbf{f}'')$ to $\mathbf{g}=\mathbf{B}\cdot\mathbf{t}'+\mathbf{f}'' \bmod q$. The combination of each of these steps is to map $(b,\mathbf{t},\mathbf{f})$ to $(b,Q(\mathbf{t},\mathbf{f}))$ where $Q(\mathbf{t},\mathbf{f})=\mathbf{B}\cdot\mathbf{t}+\mathbf{f} \bmod q + b\mathbf{c} \bmod q$, where $\mathbf{c}=\mathbf{B}\cdot\mathbf{s}+\mathbf{e}$

Observe that $Q(k,\cdot)$ is also collision resistant, under the LWE assumption. This is because a collision $(0,\mathbf{t}_0')$ and $(1,\mathbf{t}_1')$ reveals $\mathbf{s}'=\mathbf{t}_0-\mathbf{t}_1$, which breaks the LWE assumption under a sub-exponential modulus/noise ratio,.

These functions satisfy more properties that we will exploit in our proof, but we will introduce those properties as we go. We will now use these functions to prove the security of the dual-free version of Construction 52.

Theorem 56. *Suppose $Q(k,\cdot)$ is collision resistant using the parameters as set in Equation 1. Moreover, assume Π is an OP-PRP for $(2^{O(n)},n^{O(1)})$-decomposable permutations. Then the function $x \mapsto \Pi^{-1}(k_{\mathsf{out}},H(x)||0^{d-r})$ is collision-resistant just given the obfuscated programs $(\mathcal{P},\mathcal{P}^{-1})$.*

Proof. Suppose there exists an adversary $\mathcal{A}$ which, given $(\mathcal{P},\mathcal{P}^{-1})$ as generated by $\mathsf{Gen}(1^\lambda)$, finds a collision in the associated function $x \mapsto \Pi^{-1}(k_{\mathsf{out}},H(x)||0^{d-r})$ with non-negligible probability ϵ. Our goal is to show that we can use $\mathcal{A}$ to find collisions in the LWE-based hash function. We will do this through a sequence of hybrids.

Hyb_0: Here, $\mathcal{P},\mathcal{P}^{-1}$ are obfuscations of the programs P,P^{-1} specified in Construction 52.

The next four hybrids are used to gradually insert the LWE-based 2-to-1 function into the program P (and analogously P^{-1}). Please see Figure 8 for a visual representation. We will primarily focus on describing the program P; the program P^{-1} will be modified accordingly.

Hyb_1: Here, we choose u so that $q^u(2B)^v = 2^{r/(n-r)}$, where q,B,v are derived from u as in Equation 1. In other words, this makes the domain of the hash function Q exactly $n/(n-r)$ bits. Sample a random matrix $\mathbf{B} \leftarrow \mathbb{Z}_q^{v\times u}$, $\mathbf{s} \leftarrow \mathbb{Z}_q^u$ and the entries of $\mathbf{e} \in \mathbb{Z}_q^m$ are sampled from discrete Gaussians of width σ.

We divide the output of $\Pi(k_{\mathsf{in}},\cdot)$ in two different ways. The first, just like in Construction 52, is to divide the output into functions $H(\cdot)$ and $J(\cdot)$ with ranges $\{0,1\}^{n-r}$ and $\{0,1\}^r$, respectively. We denote by w the outputs of $H(x)$, and by $\mathbf{z}$ the outputs of J.

The second way is to divide into $n-r$ slices, each slice being $n/(n-r)$ bits. Each slice will correspond to an input to Q. 1 bit of each slice will belong to $\mathbf{z}=J(x)$, which we will denote by

b_i. The remaining $r/(n-r)$ bits of each slice will belong to $w = H(x)$, which we will denote by w_i.

Recall that the w_i part of each slice is interpreted as a vector $(\mathbf{t}_i, \mathbf{f}_i)$ where $\mathbf{t}_i \in \mathbb{Z}_q^u$ and $\mathbf{f}_i \in (-B, B]^v$. For now, we will set $(\mathbf{t}'_i, \mathbf{f}'_i) = (\mathbf{t}_i, \mathbf{f}_i)$, though this will change in later hybrids. We say that $(\mathbf{t}'_i, \mathbf{f}'_i)$ is "bad" if $\mathbf{f} - \mathbf{e} \notin (-B, B]^v$.

We now change the program P that gets obfuscated to $P_1(x)$, which is identical to $P(x)$ *except* for the following. On input x, let $w'_i \in \{0,1\}$ be defined as $w'_i = b_i$ if $w_i = H_i(x)$ is bad, and otherwise $w'_i = 0$. We then assemble the w'_i into the string $w' \in \{0,1\}^{n-r}$. Then P_1 computes y as $\Pi^{-1}(k_{\mathsf{out}},\ w||w'||0^{d-n}\ )$. We will likewise change P^{-1} to the program P_1^{-1}, which computes $\Pi^{-1}(k_{\mathsf{out}}, y) = w||w'||g$, computes $\mathbf{z}$ such that $\mathbf{A}(y) \cdot \mathbf{z} + \mathbf{b}(y) = \mathbf{u}$. Then it aborts and outputs $\perp$ if either (0) $g \neq 0^{d-n}$, (1) no such $\mathbf{z}$ exists, or (2) w' is not computed correctly from w and $\mathbf{z}$. Only if it does not abort does P_1 compute $x \leftarrow \Pi^{-1}(k_{\mathsf{in}}, (w, \mathbf{z}))$ and outputs x. We then have that P_1^{-1} is the correct inverse of P_1.

Observe that once we fix $\mathbf{e}$, the "bad" sets are just the union tests for whether a given coordinate of w_i is in a particular interval. These tests, being sandwiched between $\Pi(k_{\mathsf{in}}, \cdot)$ and $\Pi^{-1}(k_{\mathsf{out}}, \cdot)$ fit the form of Lemma 43, and so Hybrid 0 and Hybrid 1 are computationally indistinguishable.

Hyb_2: Let π be the permutation on each slice mapping $(b_i, w_i) = (b_i, \mathbf{t}_i, \mathbf{f}_i) \mapsto (b_i, \mathbf{t}'_i = \mathbf{t}_i + b_i\mathbf{s} \bmod q, \mathbf{f}'_i = \mathbf{f} + b_i\mathbf{e} \bmod (-B, B])$, where $h \bmod (-B, B]$ means the unique integer $h' \in (-B, B]$ such that $h - h'$ is a multiple of $2B$. We extend this to a permutation over over $\{0,1\}^n$ by applying it separately to each slice of $n/(n-r)$ bits separately.

We now switch from P_1 the program P_2 which applies the permutation π to $(\mathbf{z}, w) = \Pi(k_{\mathsf{in}}, x)$, before testing for bad-ness. Likewise, we switch from P_1^{-1} to P_2^{-1}, which applies π^{-1} to the input to $\Pi^{-1}(k_{\mathsf{out}}, \cdot)$, *after* testing for badness. By applying Lemma 40 to k_{out}, we have that Hybrid 2 is indistinguishable from Hybrid 1.

Now we see that, if the ith slice of the output of $\Pi(k_{\mathsf{in}}, x)$ is $(b_i, \mathbf{t}_i, \mathbf{f}_i)$, then the ith slice of the input to to $\Pi^{-1}(k_{\mathsf{out}}, \cdot)$ (including the bits of w') is $(\mathbf{t}'_i, \mathbf{f}'_i, w'_i)$ where

$$
\begin{aligned}
(\mathbf{t}'_i, \mathbf{f}'_i, w'_i) &= (\mathbf{t} + b_i\mathbf{s} \bmod q, \mathbf{f} + b_i\mathbf{e} \bmod (-B, B], 0) \\
&= (\mathbf{t} + b_i\mathbf{s} \bmod q, \mathbf{f} + b_i\mathbf{e}, 0) && \text{if } \mathbf{f} + b_i\mathbf{e} \bmod (-B, B] \text{ is good} \\
(\mathbf{t}'_i, \mathbf{f}'_i, w'_i) &= (\mathbf{t}_i, \mathbf{f}_i, 0) \\
&= (\mathbf{t}_i + b_i\mathbf{s} \bmod q, \mathbf{f}_i + b_i\mathbf{e}, 0) && \text{if } b_i = 0 \text{ and } \mathbf{f}_i = \mathbf{f}_i + b_i\mathbf{e} \bmod (-B, B] \text{ is bad} \\
(\mathbf{t}'_i, \mathbf{f}'_i, w'_i) &= (\mathbf{t}_i + \mathbf{s} \bmod q, \mathbf{f}_i + \mathbf{e} \bmod (-B, B], 1) && \text{if } b_i = 1 \text{ and } \mathbf{f}_i + \mathbf{e} \bmod (-B, B] \text{ is bad}
\end{aligned}
$$

In particular, the two cases where the w'_i bit is 0 actually have $\mathbf{f}_i + b_i\mathbf{e}$ over the integers and not reduced.

Hyb_3: Let $(\mathbf{t}'_i, \mathbf{f}'_i, w'_i)$ denote the contents of the ith slice of the input to $\Pi^{-1}(k_{\mathsf{out}}, \cdot)$ in Hybrid 2, after applying both the permutation π and the sparse trigger from Hybrid 1. Notice that, while the inputs are defined for any $\mathbf{t}'_i, \mathbf{f}'_i, w'_i$, the only cases arising in our program will have $w'_i = 1$ if and only if both $b_i = 1$ and $\mathbf{f}'_i$ is bad. We will call these inputs "valid".

Now let τ be the function which, on each slice $(\mathbf{t}'_i, \mathbf{g}'_i, w'_i)$ which is valid, outputs to $(\mathbf{t}'_i, \mathbf{f}''_i = [\ \mathbf{f}'_i - w'_i\mathbf{e} \bmod (-B, B]\ ] + w'_i\mathbf{e})$. Notice that the final $+w'_i\mathbf{e}$ is not reduced mod $(-B, B]$. If $w'_i = 0$, this function has no effect. If $w'_i = 1$ (which corresponds to $\mathbf{f}'_i$ being bad), then this function will map $\mathbf{f}'_i = \mathbf{f}_i + \mathbf{e} \bmod (-B, B]$ to $\mathbf{f}''_i = \mathbf{f}'_i + \mathbf{e}$ over the integers without the mod.

Next, observe that when restricted to valid inputs, τ is efficiently invertible given knowledge of $\mathbf{s}$: simply check if any of the components of $\mathbf{f}'_i - \mathbf{s}$ are outside of $(-B, B]$; this tells us the bit w'_i.

Moreover, we can simply compute $\mathbf{f}'_i = \mathbf{f}''_i \bmod (-B, B]$.

Also notice that the range of this function is $q^u \prod_{i=1}^{v}(-B - s_i, B + s_i]$ where s_i is the ith entry of $\mathbf{s}$, which we know are in $(\overline{B}, \overline{B}]$. Since B is super-polynomially larger than $\overline{B}$ (with overwhelming probability), the size of this range is at most $2 \times q^u(2B)^v$, which in turn is the size of the domain (since we are including the bit w'_i in the domain). Thus, we can interpret the range as a sub-set of the domain (when including both valid and invalid inputs in the domain).

Thus, we can actually extend τ to be a permutation on the entire slice, by arbitrarily filling in how τ behaves on invalid slices. We can then extend τ to be a permutation across all slices by permuting each slice separately.

In Hybrid 3, we therefore apply this permutation τ to the input to $\Pi^{-1}(k_{\mathsf{out}}, \cdot)$ (after applying π and performing the sparse trigger). Observe that the domain of τ (when applied to all slices) is $r + (n - r) = n$ bits. The remaining ancillas inputted to $\Pi^{-1}(k_{\mathsf{out}}, \cdot)$ are $d - n$ bits. As long as $d \geq 2n$, the number of ancillas is at least n bits. Therefore, applying the permutation τ is decomposable, meaning we can invoke Lemma 40 to see that Hybrid 3 is indistinguishable from Hybrid 2.

Since we apply τ after performing the sparse trigger, we now have that the ith slice of the input to $\Pi^{-1}(k_{\mathsf{out}}, \cdot)$ is exactly $(\mathbf{t}_i + b\mathbf{s} \bmod q, \mathbf{f}_i + b_i\mathbf{e})$ where the second term is *not* reduced. Notice that $\mathbf{f}_i + b_i\mathbf{e} \ll q/2$, so $\mathbf{f}_i + b_i\mathbf{e}$ is equivalent to $\mathbf{f}_i + b_i\mathbf{e} \bmod q$. Thus, we can write the input to $\Pi^{-1}(k_{\mathsf{out}}, \cdot)$ as $(\mathbf{t}_i + b_i\mathbf{s}, \mathbf{f}_i + b_i\mathbf{e}) \bmod q$.

Hyb_4: Now we invoke the fact that the hash function Q has a trapdoor. Namely, It is possible to sample $\mathbf{B}$ together with a trapdoor T such that, given T and $\mathbf{B} \cdot \mathbf{t} + \mathbf{f} \bmod q$ where $\mathbf{f} \in (-B - \overline{B}, B + \overline{B}]$, one can recover $\mathbf{t}, \mathbf{f}$.

Let $(\mathbf{t}'_i, \mathbf{f}''_i)$ denote the contents of the ith slice of the input to $\Pi^{-1}(k_{\mathsf{out}}, \cdot)$ in Hybrid 3, after applying π from Hybrid 2 and the sparse trigger from Hybrid 1. We will say that $(\mathbf{t}'_i, \mathbf{f}''_i)$ is valid if $\mathbf{f}''_i \in (-B - \overline{B}, B + \overline{B}]$. Let γ be the function mapping valid $(\mathbf{t}'_i, \mathbf{f}''_i)$ to $\mathbf{B} \cdot \mathbf{t}'_i + \mathbf{f}''_i \bmod q$. Notice that $\mathbf{B}$ is invertible on valid inputs given the trapdoor T. Moreover, we can extend γ to a permutation on $\mathbb{Z}_q^v$ arbitrarily. Thus, we can interpret γ as a permutation on $\log_2 q^v = v \log_2 q$ bits.

Hybrid 4 will then change P_3 into the program P_4 which applies γ to the input of $\Pi^{-1}(k_{\mathsf{out}}, \cdot)$ (after applying π from hybrid 2, the sparse trigger from Hybrid 1, and τ from Hybrid 3). As long as $d - v \log_2 q \geq v \log_2 q$, we have enough ancillas that arbitrary permutations are decomposable. Thus, Hybrid 4 and Hybrid 3 are indistinguishable.

Hyb_5: Notice that the ith slice of the input to $\Pi^{-1}(k_{\mathsf{out}}, \cdot)$ in Hybrid 4 is $\mathbf{B} \cdot \mathbf{t}'_i + \mathbf{f}''_i \bmod q = \mathbf{B} \cdot (\mathbf{t}_i + b_i\mathbf{s}) + \mathbf{f}_i + \mathbf{e} \bmod q = b_i\mathbf{c} + \mathbf{B} \cdot \mathbf{t}_i + \mathbf{f}_i \bmod q = Q((\mathbf{B}, \mathbf{c}), (b_i, \mathbf{t}_i, \mathbf{f}_i))$. Thus, in Hybrid 5, we simply change the program P_4 into the program P_5 which applies $Q((\mathbf{B}, \mathbf{c}), \cdot)$ to each slice of the output of $\Pi(k_{\mathsf{out}}, \cdot)$, and then takes the outputs of Q and feeds them into $\Pi^{-1}(k_{\mathsf{in}}, \cdot)$. This program is functionally the same as that in Hybrid 4. Thus, by iO security, Hybrids 4 and 5 are indistinguishable. See Figure 9. Notice that to compute P_4^{-1}, we need to inverse Q, which means we still need the trapdoor T.

Hyb_6: Here, we generate y exactly as in Hybrid 5. However, now we change how the output $\mathbf{u}$ is generated. We will sample a new key $k'_{\mathsf{lin}} \leftarrow \{0,1\}^\lambda$. Let $\mathbf{C}(\cdot)$ denote the first $k \times n$ bits of $\mathsf{F}(k'_{\mathsf{lin}}, \cdot)$, and $\mathbf{d}(\cdot)$ denote the next k bits.

Now let $\overline{\mathbf{w}}$ denote the output of $\Pi(k_{\mathsf{in}}, x)$ (meaning $\overline{\mathbf{w}} = (\mathbf{z}, w)$), which is interpreted as a vector in $\mathbb{Z}_2^n$. Then in the program P_6 we set $\mathbf{u} = \mathbf{C}(y) \cdot \overline{\mathbf{w}} + \mathbf{d}(y)$. See Figure 10. To compute $P_6^{-1}(y, \mathbf{u})$,

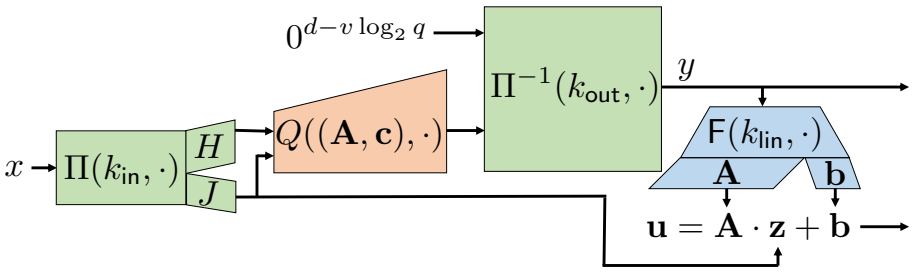

Figure 9: The program P in Hybrid 5. The output of $\Pi(k_{\mathsf{in}},\cdot)$ is divided into slices $(b_i,\mathbf{t}_i,\mathbf{f}_i)$. Now each slice of the input to $\Pi^{-1}(k_{\mathsf{out}},\cdot)$ is set to $Q((\mathbf{B},\mathbf{c}),(b_i,\mathbf{t}_i,\mathbf{f}_i))$. The bits b_i are still assembled into the vector $\mathbf{z}=J(x)$, and output $\mathbf{u}$ is a pseudorandonly-generated affine function of $\mathbf{z}$.

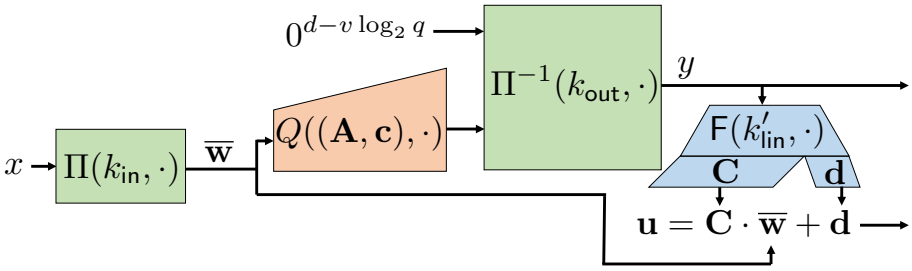

Figure 10: The program P in Hybrid 6. Now we simply apply $Q((\mathbf{B},\mathbf{c}),\cdot)$ to each slice of the output $\overline{\mathbf{w}}$ of $\Pi(k_{\mathsf{in}},\cdot)$. The output $\mathbf{u}$ is now set to a pseudorandomly-generated affine function of $\overline{\mathbf{w}}$.

we compute $(h,g)\leftarrow\Pi(k_{\mathsf{out}},y)$, where (1) the ith slice h_i of h is supposedly $Q((\mathbf{B},\mathbf{c}),\overline{w}_i)$, and (2) g is supposedly $0^{d-v\log_2 q}$. If (2) fails, abort and output $\perp$. To check (1), rather than inverting Q, we solve for $\overline{w}$ such that $\mathbf{u}=\mathbf{C}(y)\cdot\overline{w}+\mathbf{d}(y)$, and apply Q to $\overline{w}$. If (1) fails, abort and output $\perp$. Otherwise, output $x\leftarrow\Pi^{-1}(k_{\mathsf{in}},\overline{w})$.

To argue the indistinguishability of Hybrids 5 and 6, consider a slice $\overline{w}_i=(b_i,w_i)$. There are two cases:

- (b_i,w_i) is part of a collision for $Q((\mathbf{B},\mathbf{c}),\cdot)$. Notice that any collision $(b_i,w_i),(b'_i,w'_i)$ must have $b_i\neq b'_i$; we can take $b_i=0,b'_i=1$. Then the colliding set can be written as $\{\overline{w}_i,\overline{w}'_i\}=\{(1,w_i\oplus w'_i)b+(0,w_i):b\in\{0,1\}\}$, which is an affine set. Moreover, notice that the bit b corresponds exactly to b_i,b'_i.
- (b_i,w_i) is not part of a collision. Then the colliding set is just a point, which is trivially an affine set. For these slices, b_i for the pre-image set will be a fixed value.

Combining the slices together, we see that for every y, we can write the pre-image set as $\{\overline{\mathbf{A}}_y\cdot\mathbf{z}+\overline{\mathbf{b}}_y\}$, where $\mathbf{z}$ ranges over all possible $\mathbf{z}$ values for that y. $\overline{\mathbf{A}}_y$ is moreover full row-rank

Therefore, the difference between Hybrids 5 and 6 is the following:

- In Hybrid 5, $\mathbf{u}=\mathbf{A}(y)\cdot\mathbf{z}+\mathbf{b}(y)$. Recall that $(\mathbf{A}(y),\mathbf{b}(y))$ are pseudorandomly generated as $\mathsf{F}(k_{\mathsf{lin}},y)$.

- In Hybrid 6, $\mathbf{u} = \mathbf{C}(y) \cdot \overline{\mathbf{w}} + \mathbf{d}(y) = (\mathbf{C}(y) \cdot \overline{\mathbf{A}}_y) \cdot \mathbf{z} + (\mathbf{C}(y) \cdot \overline{\mathbf{b}}_y + \mathbf{d}(y))$. In other words, we are implicitly generating $\mathbf{A}(y) = \mathbf{C}(y) \cdot \overline{\mathbf{A}}_y$ and $\mathbf{b}(y) = \mathbf{C}(y) \cdot \overline{\mathbf{b}}_y + \mathbf{d}(y)$, where $(\mathbf{C}(y), \mathbf{d}(y))$ are pseudorandomly generated as $\mathsf{F}(k'_{\mathsf{lin}}, y)$.

Let D_0^y be the distribution of random $(\mathbf{A}, \mathbf{b})$, and D_1^y be the distribution $(\mathbf{A} = \mathbf{C} \cdot \overline{\mathbf{A}}, \mathbf{b} = \mathbf{C} \cdot \mathbf{b} + \mathbf{d})$, where $\mathbf{C}, \mathbf{d}$ are random. Then since each $\overline{\mathbf{A}}_y$ is assumed to be full row-rank, D_0^y and D_1^t are actually identical distributions. Thus, assuming the sub-exponential security of F and iO, Hybrids 5 and 6 are indisitnguishable by Lemma 18.

In Hybrid 6, we therefore have that $\mathcal{A}$ still finds $x_0 \neq x_1$ such that $P(x_0)$ and $P(x_1)$ output the same y. Moreover, the entire experiment can be simulated just knowing $\mathbf{A}, \mathbf{c}$. If we let $\overline{w}_0 = \Pi(k_\in, x_0)$ and $\overline{w}_1 = \Pi(k_\in, x_1)$, we see that $\overline{w}_0 \neq \overline{w}_1$ but from Figure 10 we have that $\overline{w}_0, \overline{w}_1$ collide when $Q((\mathbf{A}, \mathbf{c}), \cdot)$ is applied to each slice. Therefore, there must be some slice of $\overline{w}_0, \overline{w}_1$ that are distinct but map to the same value under $Q((\mathbf{A}, \mathbf{c}), \cdot)$. Thus, we have found a collision in $Q((\mathbf{A}, \mathbf{c}), \cdot)$. □

References

[Aar03] S. Aaronson. Quantum certificate complexity. In *18th IEEE Annual Conference on Computational Complexity, 2003. Proceedings.*, pages 171–178, 2003.

[Aar09] Scott Aaronson. Quantum copy-protection and quantum money. In *Proceedings of the 2009 24th Annual IEEE Conference on Computational Complexity*, CCC '09, page 229–242, USA, 2009. IEEE Computer Society.

[AC12] Scott Aaronson and Paul Christiano. Quantum money from hidden subspaces. In Howard J. Karloff and Toniann Pitassi, editors, *44th Annual ACM Symposium on Theory of Computing*, pages 41–60, New York, NY, USA, May 19–22, 2012. ACM Press.

[AGKZ20] Ryan Amos, Marios Georgiou, Aggelos Kiayias, and Mark Zhandry. One-shot signatures and applications to hybrid quantum/classical authentication. In Konstantin Makarychev, Yury Makarychev, Madhur Tulsiani, Gautam Kamath, and Julia Chuzhoy, editors, *52nd Annual ACM Symposium on Theory of Computing*, pages 255–268, Chicago, IL, USA, June 22–26, 2020. ACM Press.

[AGQY22] Prabhanjan Ananth, Aditya Gulati, Luowen Qian, and Henry Yuen. Pseudorandom (function-like) quantum state generators: New definitions and applications. In *Theory of Cryptography Conference*, pages 237–265. Springer, 2022.

[AMR22] Navid Alamati, Giulio Malavolta, and Ahmadreza Rahimi. Candidate trapdoor claw-free functions from group actions with applications to quantum protocols. In *Theory of Cryptography Conference*, pages 266–293. Springer, 2022.

[ARU14] Andris Ambainis, Ansis Rosmanis, and Dominique Unruh. Quantum attacks on classical proof systems: The hardness of quantum rewinding. In *55th Annual Symposium on Foundations of Computer Science*, pages 474–483, Philadelphia, PA, USA, October 18–21, 2014. IEEE Computer Society Press.

[AS04] Scott Aaronson and Yaoyun Shi. Quantum lower bounds for the collision and the element distinctness problems. *J. ACM*, 51(4):595–605, July 2004.

[AS15] Gilad Asharov and Gil Segev. Limits on the power of indistinguishability obfuscation and functional encryption. In Venkatesan Guruswami, editor, *56th Annual Symposium on Foundations of Computer Science*, pages 191–209, Berkeley, CA, USA, October 17–20, 2015. IEEE Computer Society Press.

[AS16] Gilad Asharov and Gil Segev. On constructing one-way permutations from indistinguishability obfuscation. In Eyal Kushilevitz and Tal Malkin, editors, *TCC 2016-A: 13th Theory of Cryptography Conference, Part II*, volume 9563 of *Lecture Notes in Computer Science*, pages 512–541, Tel Aviv, Israel, January 10–13, 2016. Springer Berlin Heidelberg, Germany.

[Bac90] Eric Bach. Explicit bounds for primality testing and related problems. *Mathematics of Computation*, 55:355–380, 1990.

[Bar23] James Bartusek. Personal communication, and also announced at the ntt research quantum money workshop, 2023.

[BCM+18] Zvika Brakerski, Paul Christiano, Urmila Mahadev, Umesh V. Vazirani, and Thomas Vidick. A cryptographic test of quantumness and certifiable randomness from a single quantum device. In Mikkel Thorup, editor, *59th Annual Symposium on Foundations of Computer Science*, pages 320–331, Paris, France, October 7–9, 2018. IEEE Computer Society Press.

[BGI+01] Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil P. Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. In Joe Kilian, editor, *Advances in Cryptology – CRYPTO 2001*, volume 2139 of *Lecture Notes in Computer Science*, pages 1–18, Santa Barbara, CA, USA, August 19–23, 2001. Springer Berlin Heidelberg, Germany.

[BGI14] Elette Boyle, Shafi Goldwasser, and Ioana Ivan. Functional signatures and pseudorandom functions. In Hugo Krawczyk, editor, *PKC 2014: 17th International Conference on Theory and Practice of Public Key Cryptography*, volume 8383 of *Lecture Notes in Computer Science*, pages 501–519, Buenos Aires, Argentina, March 26–28, 2014. Springer Berlin Heidelberg, Germany.

[BGK+24] James Bartusek, Vipul Goyal, Dakshita Khurana, Giulio Malavolta, Justin Raizes, and Bhaskar Roberts. Software with certified deletion. In Marc Joye and Gregor Leander, editors, *Advances in Cryptology – EUROCRYPT 2024, Part IV*, volume 14654 of *Lecture Notes in Computer Science*, pages 85–111, Zurich, Switzerland, May 26–30, 2024. Springer, Cham, Switzerland.

[BK23] James Bartusek and Dakshita Khurana. Cryptography with certified deletion. In Helena Handschuh and Anna Lysyanskaya, editors, *Advances in Cryptology – CRYPTO 2023, Part V*, volume 14085 of *Lecture Notes in Computer Science*, pages 192–223, Santa Barbara, CA, USA, August 20–24, 2023. Springer, Cham, Switzerland.

[BKP23] James Bartusek, Dakshita Khurana, and Alexander Poremba. Publicly-verifiable deletion via target-collapsing functions. In Helena Handschuh and Anna Lysyanskaya, editors, *Advances in Cryptology – CRYPTO 2023, Part V*, volume 14085 of *Lecture Notes in Computer Science*, pages 99–128, Santa Barbara, CA, USA, August 20–24, 2023. Springer, Cham, Switzerland.

[BKW17] Dan Boneh, Sam Kim, and David J. Wu. Constrained keys for invertible pseudorandom functions. In Yael Kalai and Leonid Reyzin, editors, *TCC 2017: 15th Theory of Cryptography Conference, Part I*, volume 10677 of *Lecture Notes in Computer Science*, pages 237–263, Baltimore, MD, USA, November 12–15, 2017. Springer, Cham, Switzerland.

[BLVW19] Zvika Brakerski, Vadim Lyubashevsky, Vinod Vaikuntanathan, and Daniel Wichs. Worst-case hardness for LPN and cryptographic hashing via code smoothing. In Yuval Ishai and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2019, Part III*, volume 11478 of *Lecture Notes in Computer Science*, pages 619–635, Darmstadt, Germany, May 19–23, 2019. Springer, Cham, Switzerland.

[BPW16] Nir Bitansky, Omer Paneth, and Daniel Wichs. Perfect structure on the edge of chaos - trapdoor permutations from indistinguishability obfuscation. In Eyal Kushilevitz and Tal Malkin, editors, *TCC 2016-A: 13th Theory of Cryptography Conference, Part I*, volume 9562 of *Lecture Notes in Computer Science*, pages 474–502, Tel Aviv, Israel, January 10–13, 2016. Springer Berlin Heidelberg, Germany.

[BW13] Dan Boneh and Brent Waters. Constrained pseudorandom functions and their applications. In Kazue Sako and Palash Sarkar, editors, *Advances in Cryptology – ASIACRYPT 2013, Part II*, volume 8270 of *Lecture Notes in Computer Science*, pages 280–300, Bengalore, India, December 1–5, 2013. Springer Berlin Heidelberg, Germany.

[CL18] Ran Canetti and Amit Lichtenberg. Certifying trapdoor permutations, revisited. In Amos Beimel and Stefan Dziembowski, editors, *TCC 2018: 16th Theory of Cryptography Conference, Part I*, volume 11239 of *Lecture Notes in Computer Science*, pages 476–506, Panaji, India, November 11–14, 2018. Springer, Cham, Switzerland.

[CLG09] Denis X. Charles, Kristin E. Lauter, and Eyal Z. Goren. Cryptographic hash functions from expander graphs. *Journal of Cryptology*, 22(1):93–113, 2009.

[CLTV15] Ran Canetti, Huijia Lin, Stefano Tessaro, and Vinod Vaikuntanathan. Obfuscation of probabilistic circuits and applications. In Yevgeniy Dodis and Jesper Buus Nielsen, editors, *TCC 2015: 12th Theory of Cryptography Conference, Part II*, volume 9015 of *Lecture Notes in Computer Science*, pages 468–497, Warsaw, Poland, March 23–25, 2015. Springer Berlin Heidelberg, Germany.

[CS20] Andrea Coladangelo and Or Sattath. A quantum money solution to the blockchain scalability problem. *Quantum*, 4:297, 2020.

[DH76] Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, 1976.

[Dra23] Justin Drake. One-shot signatures. talk given at the programmable cryptography conference progcrypto, 2023. https://www.youtube.com/watch?v=VmqkH3NPG_s.

[DS23] Marcel Dall'Agnol and Nicholas Spooner. On the Necessity of Collapsing for Post-Quantum and Quantum Commitments. In Omar Fawzi and Michael Walter, editors, *18th Conference on the Theory of Quantum Computation, Communication and Cryptography (TQC 2023)*, volume 266 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 2:1–2:23, Dagstuhl, Germany, 2023. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

[FGH+12] Edward Farhi, David Gosset, Avinatan Hassidim, Andrew Lutomirski, and Peter W. Shor. Quantum money from knots. In Shafi Goldwasser, editor, *ITCS 2012: 3rd Innovations in Theoretical Computer Science*, pages 276–289, Cambridge, MA, USA, January 8–10, 2012. Association for Computing Machinery.

[GGH+13] Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. In *54th Annual Symposium on Foundations of Computer Science*, pages 40–49, Berkeley, CA, USA, October 26–29, 2013. IEEE Computer Society Press.

[GP07] Louis Granboulan and Thomas Pornin. Perfect block ciphers with small blocks. In Alex Biryukov, editor, *Fast Software Encryption – FSE 2007*, volume 4593 of *Lecture Notes in Computer Science*, pages 452–465, Luxembourg, Luxembourg, March 26–28, 2007. Springer Berlin Heidelberg, Germany.

[GPSZ17] Sanjam Garg, Omkant Pandey, Akshayaram Srinivasan, and Mark Zhandry. Breaking the sub-exponential barrier in obfustopia. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *Advances in Cryptology – EUROCRYPT 2017, Part III*, volume 10212 of *Lecture Notes in Computer Science*, pages 156–181, Paris, France, April 30 – May 4, 2017. Springer, Cham, Switzerland.

[GR13] Oded Goldreich and Ron D. Rothblum. Enhancements of trapdoor permutations. *Journal of Cryptology*, 26(3):484–512, 2013.

[HMR12] Viet Tung Hoang, Ben Morris, and Phillip Rogaway. An enciphering scheme based on a card shuffle. In Reihaneh Safavi-Naini and Ran Canetti, editors, *Advances in Cryptology – CRYPTO 2012*, volume 7417 of *Lecture Notes in Computer Science*, pages 1–13, Santa Barbara, CA, USA, August 19–23, 2012. Springer Berlin Heidelberg, Germany.

[HPPY24] Alexander Hoover, Sarvar Patel, Giuseppe Persiano, and Kevin Yeo. Plinko: Single-server PIR with efficient updates via invertible PRFs. Cryptology ePrint Archive, Report 2024/318, 2024.

[KNY23] Fuyuki Kitagawa, Ryo Nishimaki, and Takashi Yamakawa. Publicly verifiable deletion from minimal assumptions. In Guy N. Rothblum and Hoeteck Wee, editors, *TCC 2023: 21st Theory of Cryptography Conference, Part IV*, volume 14372 of *Lecture Notes in Computer Science*, pages 228–245, Taipei, Taiwan, November 29 – December 2, 2023. Springer, Cham, Switzerland.

[KPTZ13] Aggelos Kiayias, Stavros Papadopoulos, Nikos Triandopoulos, and Thomas Zacharias. Delegatable pseudorandom functions and applications. In Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors, *ACM CCS 2013: 20th Conference on Computer and*

Communications Security, pages 669–684, Berlin, Germany, November 4–8, 2013. ACM Press.

[KSS22] Daniel M. Kane, Shahed Sharif, and Alice Silverberg. Quantum money from quaternion algebras. *Mathematical Cryptology*, 2(1):60–83, Oct. 2022.

[LLQ22] Jiahui Liu, Qipeng Liu, and Luowen Qian. Beating classical impossibility of position verification. In Mark Braverman, editor, *ITCS 2022: 13th Innovations in Theoretical Computer Science Conference*, volume 215, pages 100:1–100:11, Berkeley, CA, USA, January 31 – February 3, 2022. Leibniz International Proceedings in Informatics (LIPIcs).

[LMZ23] Jiahui Liu, Hart Montgomery, and Mark Zhandry. Another round of breaking and making quantum money: How to not build it from lattices, and more. In Carmit Hazay and Martijn Stam, editors, *Advances in Cryptology – EUROCRYPT 2023, Part I*, volume 14004 of *Lecture Notes in Computer Science*, pages 611–638, Lyon, France, April 23–27, 2023. Springer, Cham, Switzerland.

[Mor05] Ben Morris. The mixing time of the Thorp shuffle. In Harold N. Gabow and Ronald Fagin, editors, *37th Annual ACM Symposium on Theory of Computing*, pages 403–412, Baltimore, MA, USA, May 22–24, 2005. ACM Press.

[MR14] Ben Morris and Phillip Rogaway. Sometimes-recurse shuffle - almost-random permutations in logarithmic expected time. In Phong Q. Nguyen and Elisabeth Oswald, editors, *Advances in Cryptology – EUROCRYPT 2014*, volume 8441 of *Lecture Notes in Computer Science*, pages 311–326, Copenhagen, Denmark, May 11–15, 2014. Springer Berlin Heidelberg, Germany.

[MY23] Tomoyuki Morimae and Takashi Yamakawa. Proofs of quantumness from trapdoor permutations. In Yael Tauman Kalai, editor, *ITCS 2023: 14th Innovations in Theoretical Computer Science Conference*, volume 251, pages 87:1–87:14, Cambridge, MA, USA, January 10–13, 2023. Leibniz International Proceedings in Informatics (LIPIcs).

[MZRA22] Yiping Ma, Ke Zhong, Tal Rabin, and Sebastian Angel. Incremental Offline/Online PIR. In *31st USENIX Security Symposium (USENIX Security 22)*, pages 1741–1758, Boston, MA, August 2022. USENIX Association.

[NTT23] NTT Research Quantum Money Workshop, 2023.

[Pat01] Jacques Patarin. Generic attacks on Feistel schemes. In Colin Boyd, editor, *Advances in Cryptology – ASIACRYPT 2001*, volume 2248 of *Lecture Notes in Computer Science*, pages 222–238, Gold Coast, Australia, December 9–13, 2001. Springer Berlin Heidelberg, Germany.

[QSi24] QSig Workshop, 2024. https://informatics.ed.ac.uk/blockchain/events/previous-events/qsig.

[RY13] Thomas Ristenpart and Scott Yilek. The mix-and-cut shuffle: Small-domain encryption secure against N queries. In Ran Canetti and Juan A. Garay, editors, *Advances in Cryptology – CRYPTO 2013, Part I*, volume 8042 of *Lecture Notes in Computer Science*,

pages 392–409, Santa Barbara, CA, USA, August 18–22, 2013. Springer Berlin Heidelberg, Germany.

[SACM21] Elaine Shi, Waqar Aqeel, Balakrishnan Chandrasekaran, and Bruce M. Maggs. Puncturable pseudorandom sets and private information retrieval with near-optimal online bandwidth and time. In Tal Malkin and Chris Peikert, editors, *Advances in Cryptology – CRYPTO 2021, Part IV*, volume 12828 of *Lecture Notes in Computer Science*, pages 641–669, Virtual Event, August 16–20, 2021. Springer, Cham, Switzerland.

[Sat22] Or Sattath. Quantum prudent contracts with applications to bitcoin, 2022.

[Shm22a] Omri Shmueli. Public-key quantum money with a classical bank. In Stefano Leonardi and Anupam Gupta, editors, *54th Annual ACM Symposium on Theory of Computing*, pages 790–803, Rome, Italy, June 20–24, 2022. ACM Press.

[Shm22b] Omri Shmueli. Semi-quantum tokenized signatures. In Yevgeniy Dodis and Thomas Shrimpton, editors, *Advances in Cryptology – CRYPTO 2022, Part I*, volume 13507 of *Lecture Notes in Computer Science*, pages 296–319, Santa Barbara, CA, USA, August 15–18, 2022. Springer, Cham, Switzerland.

[Sho94] Peter W. Shor. Algorithms for quantum computation: Discrete logarithms and factoring. In *35th Annual Symposium on Foundations of Computer Science*, pages 124–134, Santa Fe, NM, USA, November 20–22, 1994. IEEE Computer Society Press.

[SW14] Amit Sahai and Brent Waters. How to use indistinguishability obfuscation: deniable encryption, and more. In David B. Shmoys, editor, *46th Annual ACM Symposium on Theory of Computing*, pages 475–484, New York, NY, USA, May 31 – June 3, 2014. ACM Press.

[Unr16a] Dominique Unruh. Collapse-binding quantum commitments without random oracles. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *Advances in Cryptology – ASIACRYPT 2016, Part II*, volume 10032 of *Lecture Notes in Computer Science*, pages 166–195, Hanoi, Vietnam, December 4–8, 2016. Springer Berlin Heidelberg, Germany.

[Unr16b] Dominique Unruh. Computationally binding quantum commitments. In Marc Fischlin and Jean-Sébastien Coron, editors, *Advances in Cryptology – EUROCRYPT 2016, Part II*, volume 9666 of *Lecture Notes in Computer Science*, pages 497–527, Vienna, Austria, May 8–12, 2016. Springer Berlin Heidelberg, Germany.

[Wie83] Stephen Wiesner. Conjugate coding. *SIGACT News*, 15(1):78–88, January 1983.

[Yao82] Andrew Chi-Chih Yao. Theory and applications of trapdoor functions (extended abstract). In *23rd Annual Symposium on Foundations of Computer Science*, pages 80–91, Chicago, Illinois, November 3–5, 1982. IEEE Computer Society Press.

[YZW+19] Yu Yu, Jiang Zhang, Jian Weng, Chun Guo, and Xiangxue Li. Collision resistant hashing from sub-exponential learning parity with noise. In Steven D. Galbraith and Shiho Moriai, editors, *Advances in Cryptology – ASIACRYPT 2019, Part II*, volume 11922 of *Lecture Notes in Computer Science*, pages 3–24, Kobe, Japan, December 8–12, 2019. Springer, Cham, Switzerland.

[Zha15] Mark Zhandry. A note on the quantum collision and set equality problems. *Quantum Info. Comput.*, 15(7–8):557–567, May 2015.

[Zha16] Mark Zhandry. A note on quantum-secure prps. *arXiv preprint arXiv:1611.05564*, 2016.

[Zha19] Mark Zhandry. Quantum lightning never strikes the same state twice. In Yuval Ishai and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2019, Part III*, volume 11478 of *Lecture Notes in Computer Science*, pages 408–438, Darmstadt, Germany, May 19–23, 2019. Springer, Cham, Switzerland.

[Zha21] Mark Zhandry. How to construct quantum random functions. *Journal of the ACM (JACM)*, 68(5):1–43, 2021.

[Zha22] Mark Zhandry. New constructions of collapsing hashes. In Yevgeniy Dodis and Thomas Shrimpton, editors, *Advances in Cryptology – CRYPTO 2022, Part III*, volume 13509 of *Lecture Notes in Computer Science*, pages 596–624, Santa Barbara, CA, USA, August 15–18, 2022. Springer, Cham, Switzerland.

[Zha24] Mark Zhandry. Quantum money from abelian group actions. In Venkatesan Guruswami, editor, *ITCS 2024: 15th Innovations in Theoretical Computer Science Conference*, volume 287, pages 101:1–101:23, Berkeley, CA, USA, January 30 – February 2, 2024. Leibniz International Proceedings in Informatics (LIPIcs).