Lower Bounds on Assumptions Behind Registration-Based Encryption

Mohammad Hajiabadi[1], Mohammad Mahmoody[2], Wei Qi[2], and Sara Sarfaraz[1]

[1] University of Waterloo
[2] University of Virginia

Abstract. Registration-based encryption (RBE) [GHMR18] is a primitive that aims to offer what identity-based encryption (IBE) [BF01] offers without the so-called key-escrow problem. In RBE parties who wish to join the system will generate their own secret and public keys and register their public keys to a transparent party called key curator (KC) who does not have any secret state.

The initial constructions of RBE made *non-black-box* use of building block primitives, due to their use of either indistinguishability obfuscation [GHMR18] or some garbling scheme [GHM+19]. More recently, it was shown [GKMR22, HLWW23] how to achieve *black-box* constructions of (variants of) RBE and even stronger primitives based on *bilinear maps* in which the RBE is relaxed to have a CRS whose length can *grow* with the number of registered identities. Making cryptographic constructions in general, and RBE in particular, black-box is an important step as it can play a significant role in its efficiency and potential deployment. Hence, in this work we ask: *what are the minimal assumptions for black-box constructions of RBE?* Particularly, can we black-box construct RBE schemes from the same assumptions used for public-key encryption or simpler algebraic assumptions that hold in the generic group model?

In this work, we prove the first black-box separation results for RBE beyond the separations that follow from the observation that RBE black-box implies public-key encryption. In particular, we answer both of the questions above negatively and prove that neither trapdoor permutations nor (even Shoup's) generic group model can be used as the sole source of hardness for building RBE schemes. More generally, we prove that a relaxation of RBE in which all the keys are registered and compressed at the same time is already too complex to be built from either of the above-mentioned primitives in a black-box way. At a technical level, using compression techniques, we prove lemmas in the TDP and GGM oracle settings that prove the following intuitive yet useful fact: that compact strings cannot signal too many trapdoors, even if their generation algorithm takes exponential time. Due to their generality, our lemmas could be of independent interest and find more applications.

Keywords: Registration-based encryption · Black-box separations · Trapdoor permutations · Generic group model.

Table of Contents

1 Introduction

Registration-based encryption (RBE) [GHMR18] is a primitive that aims to offer what identity-based encryption (IBE) [BF01] offers while avoiding the key-escrow problem. Indeed, IBE suffers from the fact that the third party, (i.e., the private-key generator,) has a universal trapdoor, (i.e., the master secret key,) allowing it to decrypt all messages encrypted using the public parameter. However, in RBE, identities generate their own secret and public keys and then simply *register* their public keys to a transparent (public-state) entity, called the *key curator* (KC). All KC does is accumulating and compressing the registered public keys in an online fashion as more and more identities register in the system. Several works were proposed to add even more desirable properties to RBE. [GHM+19] showed how to base RBE on standard assumptions. [GV20] constructed verifiable RBE, where the KC can give succinct proof for the existence/non-existence of users in the system. In [WLW+23], blockchain was used to construct transparent RBE, making the system even more decentralized by letting the individual participants instead of the KC manage the keys. Inspired by RBE, more advanced primitives were defined and constructed, including registered attribute based encryption (ABE) [HLWW23] and registered functional encryption (FE) [DP23, FFM+23].

The initial constructions of RBE were all *non-black-box*, in the sense that implementing them required knowing the implementation details of *at least* one of the primitives that were used in those constructions. This was due to use of code obfuscation [GHMR18] and/or garbled circuits [GHM+19] in such constructions. This was an undesired state of affair, as non-black-box constructions are more likely to be inefficient for practical use, and indeed RBE is yet to become

practical enough for real world deployment. This is in contrast with the closely-related primitive of IBE for which *black-box* constructions from pairing-based assumptions on bilinear maps exist [BF01, BB04].

More recently, the works of [GKMR22, HLWW23] showed how to achieve *black-box* constructions of RBE, provided that it is *relaxed* so that the common reference string (CRS) can grow as a function of the total number of registered identities.[3] These works suggest that perhaps even standard RBE could at some point be realized based on well-founded assumptions in a *black-box* manner. In the meantime, in this work, we focus on a tightly related question: if we could base RBE on black-box assumptions, how simple those assumptions need to be? In particular, *what are the black-box minimal assumptions for constructing RBE*? To answer this question, we also need to understand the *black-box barriers* that arise against building RBE from the *more desirable* type assumptions. In particular, prior to our work, it was not known whether RBE can be solely based on the assumption that public-key encryption (PKE) exists.[4] Moreover, it was not known whether simpler algebraic assumptions than those on bilinear maps (e.g., assumptions that hold in the generic group model) may suffice for RBE in a black-box way. We emphasize that although it is known how to bootstrap RBE to IBE [BLSV18, DG17], black-box separations known for IBE [BPR+08] do not automatically carry over to RBE, as the bootstrapping is non-black-box.

1.1 Our Results

One approach to study the black-box complexity of RBE is to study the possibility of constructing it in idealized models that provide PKE or simple algebraic assumptions for free. In particular, the random trapdoor permutation (TDP) oracle model provides (CCA-secure) PKE (among other primitives such as collision-resistant hashing) as a black-box. Moreover, the generic group model (GGM) [Sho97, Mau05] is an idealized model in which assumptions such as hardness of discrete logarithm, CDH (computational Diffie-Hellman assumption), and even DDH (*decisional* Diffie-Hellman assumption) hold. Hence, it is a very natural model for studying the possibility of realizing RBE from those assumptions, which is the approach that we pursue as well.

In this work, we prove the first non-trivial black-box separations about the complexity of RBE. In particular, we prove the following theorem.

Theorem 1 (Main results – informal). *There is no black-box construction of RBEs in either of the idealized models of random trapdoor permutations (TDPs) or Shoup's generic group model. Our impossibility results hold even if the CRS in RBE can grow (polynomially) with the number of registered identities.*

[3] The work of [HLWW23] further generalizes the primitive to *attribute-based* encryption and constructs registered ABE, while further relaxing the primitive and allowing *interactive* registration.

[4] Note that PKE is indeed necessary for RBE in a black-box way.

In particular, what we prove is that there is no construction of RBEs whose security is *solely* based on the idealized models stated in Theorem 1. We do so by proving that such schemes can be broken by a polynomial-query attacker. This is sufficient for ruling out a fully black-box construction [RTV04]. Ruling out constructions of RBE in the idealized model of TDPs would also rule out using any primitive $\mathcal{P}$ (or rather sets of primitives) that can be constructed from TDP oracles in a black-box way (e.g., collision-resistant hash functions and public-key encryption). Ruling out RBE in the GGM would also prove a black-box separation from concrete assumptions that hold in this model (e.g., DDH).

In Section 2, we give an in-depth technical overview of the proof of Theorem 1. However, here we present some high level discussions about Theorem 1 and why it does not follow from previous separations known for IBE.

Public-key compression. We prove Theorem 1 by demonstrating a more general result about a primitive that is *weaker* than RBE and yet is black-box implied by RBE; we refer to that primitive as *public-key compression* (or PKCom, for short). PKCom can be thought of as a *static* variant of RBE. In PKCom a polynomial number of identities who have generated their own public and secret keys all arrive *at the same time*. Then, they send their public-keys $\mathsf{pk}_1, \ldots, \mathsf{pk}_n$ to be compressed by the key curator (KC) in "one shot". Then, the compressed public parameter pp is broadcast by the KC to the registered identities and can be used similarly to the public parameter of IBE to privately encrypt messages to registered identities. When it comes to decryption, all parties will have access to each other's *public* keys as well as their own secret key, but of course they do not have access to each others' secret keys. A very closely related primitive to PKCom (called *slotted registered* ABE, as a generalization of IBE) is also introduced in [HLWW23] and is shown to be black-box equivalent to RBE. However, our primitive is still weaker, as it works with the fixed identities set $\{1, 2, \ldots\}$ rather than arbitrary strings. These limitations make our impossibility result stronger.

Main challenge for proving separations for RBE. By now, multiple black-box separations are known for assumptions behind IBE. Boneh et al. [BPR+08] proved that IBE cannot be black-box constructed from random trapdoor permutations, and the works of [PRV12, Zha22] showed that IBE cannot be built in what is known as *Shoup's* generic group model [Sho97].[5] However, we emphasize that, none of these works imply a separation for *registration* based encryption (and its relaxation PKCom). Below, we first describe the similarities between the two settings, and then we describe the major difference between them.

The core reason underlying both impossibilities for IBE and ours for RBE is that a compact public parameter (pp) cannot encode enough information for securely encrypting to more than an a priori bounded number of identities. However, when it comes to RBE, there is a big difference that makes the previous IBE

[5] More specifically, [PRV12] claimed the result in a model that is a mixture of Maurer's [Mau05] and Shoup's [Sho97] models. Then, [SGS21] proved (a tight) separation in Murer's model, and finally, [Zha22] proved the separation of IBE in Shoup's model.

separation techniques come short. The IBE separation proofs [BPR+08, Zha22] proceed by crucially relying on the bounded *running time* of the setup algorithm: in an IBE scheme, the setup algorithm, which generates $(\mathsf{pp}, \mathsf{msk})$, makes a *fixed* polynomial q number of $(\mathsf{pk}, \mathsf{sk})$ (trapdoor) queries to its oracle. So, if one corrupts a sufficiently larger-than-q number of random identities and extracts their dedicated trapdoors (say by repeated decryptions), the attacker will then recover all the needed trapdoors and can decrypt the challenge ciphertext of a non-corrupted identity as well. This "combinatorial" argument, however, completely breaks down when we move to RBE. Indeed the *running time* of generating a public parameter of an n-user RBE *grows* with the parameter n itself, because this public parameter is generated through n registration steps. Therefore, the "setup" procedure (as analogous to IBE) that generates the public parameter might be asking n or more trapdoor queries.

Compression techniques. To get around the above challenge, we introduce new compression techniques to prove exactly that a short pp still cannot encode enough information for all n users, *regardless* of how much time it takes to generate it (see more on this in the technical overview). In fact, our new compression tool, stated in Lemma 2, (i.e., a bounded-size message/pp cannot compress too many trapdoors no matter how long it takes to generate it) proves a basic and intuitive fact that could potentially find other applications beyond RBE. For starters, it already allows us to rule out black-box construction of "leveled" IBE, where the number of users n is known during setup and the setup time is allowed to grow with n while the pp is compact, from the GGM. None of the previous IBE separation techniques allows for proving such an impossibility.

Shoup's GGM vs. Maurer's GGM. In Shoup's GGM [Sho97] group elements do have (random-looking) representations. Therefore, impossibility results in this model imply further black-box separations (e.g., from public-key encryption). However, these corollaries do not automatically follow from results that the primitive is impossible in Maurer's model [Mau05], in which group operations are outsourced to a black-box oracle. In fact, proving the impossibility of a primitive X in Maurer's model does not even imply that X is black-box impossible from PKE. In particular, some impossibility results in Maurer's GGM *cannot* be extended to Shoup's model; e.g., [RSS20] ruled out sequential (delay) functions in Maurer's generic group, while such functions can be obtained from random oracles, which in turn can be obtained in Shoup's model. As another example, there exists natural DDH-based constructions of primitives such as rate-1 OT and private-information retrieval (PIR) in Shoup's model [DGI+19], yet these primitives can be proved impossible in Maurer's group. Thus, proving an impossibility in Shoup's model gives the stronger and more meaningful impossibility. See [Zha22] for more discussion on this topic.

Limits of RBE in Maurer's GGM. For the reasons above, in this work we aim for proving impossibility of RBE (and PKCom) in Shoup's GGM. Having said that, we first show that a separation of RBE/PKCom in Maurer's GGM *can*

indeed be proved as well. Our starting point for showing this barrier is the work of [SGS21] that ruled out *identity* based encryption in Maurer's GGM. Despite focusing on IBE, their proof does *not* care about the exact running time that it takes to generate the public parameter, and it only relies on the *number* of group elements that are explicitly planted in the public parameter. This makes their result general enough to be basically applicable to our PKCom as well, if one opens up their proof to adapt it to RBE. One limitation of this result, however, is that it does not allow CRS to be present, and we particularly would like to even allow our PKCom (and RBE) schemes to have CRS that can *grow* with the number of identities. Such extensions would make our impossibility result *complement* the recent positive (black-box) results of [GKMR22, HLWW23] in which hardness assumptions in pairing-based groups are used to construct RBEs with polynomially long CRS. The follow-up works of [DHH+21, CFGG23] further generalized the initial impossibility of [SGS21] to the point that we could use their results to formally obtain an impossibility of RBE from Maurer's GGM as corollary, even with polynomially long CRS. The reason is that RBEs, just like IBEs, can be used to obtain signature schemes using the so-called Naor's trick,[6] and the works of [DHH+21, CFGG23] do indeed rule out the existence of digital signatures for a polynomially large message space in Maurer's GGM, even if the scheme has a polynomially long CRS.

2 Technical Overview

We prove that public-key compression schemes cannot be built in a black-box way from TDPs or in the Shoup generic group. Since RBE and PKCom are black-box equivalent, our impossibility results will also apply to RBE.[7]

We prove our two impossibility results by showing that relative to a random TDP oracle or a GGM oracle with explicit random labels (i.e., Shoup's model [Sho97]), PKCom cannot exist so long as its security is solely based on the oracle model. More specifically, we show that any purported PKCom construction relative to either a random TDP oracle a GGM oracle can be broken by an adversary who makes a polynomial number of queries, while the adversary can do unbounded amount of computation independent of the oracle.

Outline. We follow the approach of Zhandry [Zha22] for proving our impossibility results. Zhandry proved that a special type of signature schemes, simply called *restricted* signatures, defined in an oracle model, cannot be realized relative to any oracles. Thus, to prove an impossibility of a primitive X relative to an oracle O, it suffices to show how to black-box transform any purported construction of X relative to O into a restricted signature relative to O without losing correctness and security.[8] The rest follows by the impossibility of restricted signatures.

[6] This is done by interpreting the decryption keys as signatures over the identity's names interpreted as messages.

[7] The fact that RBE black-box implies PKCom is straightforward, due to PKCom being a special case. The converse is also true and is proved in [HLWW23].

[8] By security, here we refer to security against unbounded poly-query adversaries.

A restricted signature scheme relative to an oracle O has the normal algorithms $(\mathrm{Gen}^O, \mathrm{Sig}^O, \mathrm{Ver}^O)$ of a signature scheme, but the verification algorithm Ver^O is restricted in the following way: $\mathrm{Ver}^O(\mathsf{vrk}, m, \sigma) = \mathrm{Ver1}(\mathrm{Ver0}^O(\mathsf{vrk}, m), \sigma)$, where Ver1 makes no oracle calls and vrk, m, σ denote the verification key, message and signature, respectively. That is, the verification algorithm is restricted in that all oracle queries may be made prior to seeing the signature σ, and upon seeing σ no further queries are permitted. Zhandry proved that no restricted signatures exist relative to any oracle O by showing that any such construction can be broken by an adversary that makes a polynomial number of queries to O. As an example of a non-restricted scheme, consider Lamport's signatures from OWFs f. In that construction, σ corresponds to OWF pre-images, while vrk correspond to OWF image values. To verify a signature σ, one will check a number of image values against their corresponding pre-images by calling f, making the construction inherently non-restricted, as expected.

Zhandry proved that certain impossibility results, such as the impossibility of IBE from GGM [SGS21, PRV12, BPR+08], may be proved more naturally using the restricted signatures methodology. In particular, Zhandry showed that one can black-box transform any IBE construction $\mathsf{IBE}^{\mathsf{GGM}}$ relative to a GGM oracle into a restricted signature $\mathcal{E}^{\mathsf{GGM}}$, while preserving its correctness and security.

Target restricted signatures. For our impossibility results, we would need to further modify the notion of restricted signatures and work with a primitive which we call *target*-restricted signatures. A target restricted signature is defined over a message space $[n] = \{1, \ldots, n\}$ (think of n as the number of PKCom users), where the verification algorithm is restricted as it is in restricted signatures, but correctness and security only hold with respect to a random target message chosen as $h \leftarrow [n]$, where the verification and signing keys in turn may depend on h. That is, $\mathrm{Gen}^O(1^n, h \in [n])$ outputs a pair of keys $(\mathsf{vrk}, \mathsf{sgk})$, and we require that the following holds. (a) δ-target correctness: for a signature σ derived for the target message h as $\sigma \leftarrow \mathrm{Sig}^O(\mathsf{sgk}, h)$ we have $\mathrm{Ver}^O(\mathsf{vrk}, h, \sigma) \geq \delta$, where Ver^O is restricted as above. (b) Zero-time unforgeability: given vrk (derived based on a random $h \leftarrow [n]$) and h, no poly-query adversary can forge a signature on h (Definition 4). We show that Zhandry's proof with almost no modifications also shows that target restricted signatures for non-negligible δ's are impossible.

Transformation. After establishing the notion of target-restricted signatures, the bulk of our technical work is as follows. For a TDP/GGM oracle O, we show how to black-box transform a purported PKCom construction CMP^O into a δ-correct target restricted signatures, where δ is non-negligible. This is where our new compression techniques come into play. Below, we first go over more details of our techniques for the case of TDP oracle, as it captures most of the challenges.

Warm-up: restricted oracle calls. As a warm-up, first let us consider the case where the TDP oracles $(\mathbf{g}, \mathbf{e}, \mathbf{d})$ are used by the PKCom scheme $\mathsf{PKCom}^{\mathbf{g},\mathbf{e},\mathbf{d}}$, in a special was as $\mathsf{PKCom}^{\mathbf{g},\mathbf{e},\mathbf{d}} := (\mathsf{Key}^{\mathbf{g}}, \mathsf{Com}^{\mathbf{g}}, \mathsf{Enc}^{\mathbf{e}}, \mathsf{Dec}^{\mathbf{d}})$, where Key is the (public and secret) key generation algorithm, Com is the key compression algorithm,

Enc is the encryption algorithm, and Dec is the decryption algorithm. Moreover, assume we do not have a CRS. This setting is already non-trivial, and helps us get our main insights across. The TDP oracles are defined randomly while inducing a permutation over the message space $\{0,1\}^\kappa$ (Definition 4). Our goal is to black-box transform such $\mathsf{PKCom}^{\mathbf{g},\mathbf{e},\mathbf{d}}$ constructions into a restricted signature relative to $(\mathbf{g},\mathbf{e},\mathbf{d})$ with comparable correctness and security.

Non-restricted signatures from PKCom. A PKCom scheme over identities $[n]$ naturally induces a signature scheme using Naor's trick (that applies to IBE schemes but can be adapted to RBE as well). In that transformation, the secret keys of an identity are kept as the signature for that identity's string. The verification then proceeds by testing the quality of the decryption key (as the signature) through repeated encryption and decryptions. This scheme is clearly *not* restricted. Below, we first describe construction that is a modification of Naor's trick construction which is still *not* restricted. However, our scheme has the benefit that we can make it restricted with more modifications which we will explain below. Let $(\mathsf{Key}^{\mathbf{g}},\mathsf{Com}^{\mathbf{g}},\mathsf{Enc}^{\mathbf{e}},\mathsf{Dec}^{\mathbf{d}})$ be the purported PKCom scheme relative to a TDP oracle $(\mathbf{g},\mathbf{e},\mathbf{d})$. A verification/signature key pair $(\mathsf{vrk},\mathsf{sgk}) \leftarrow \mathrm{Gen}^O(1^n, h \in [n])$ on a target message $h \in [n]$ is obtained as follows: generate n public/secret key pairs $\{(\mathsf{pk}_i,\mathsf{sk}_i)\}_{i\in[n]}$ by running $\mathsf{Key}^{\mathbf{g}}(1^\kappa)$, and let QGen_i denote the set of query-answer (Q-A for short) pairs obtained for generating $(\mathsf{pk}_i,\mathsf{sk}_i)$. Let $\mathsf{pp} := \mathsf{Com}^{\mathbf{g}}(\mathsf{pk}_1,\ldots,\mathsf{pk}_n)$. Output $\mathsf{vrk} := (\mathsf{pp},\{\mathsf{QGen}_i\}_{i\neq h})$ and $\mathsf{sgk} := (\mathsf{sk}_h,\mathsf{QGen}_h)$.[9] A signature on h is $(\mathsf{sk}_h,\mathsf{QGen}_h)$. Define $\mathrm{Ver}^O(\mathsf{vrk},h,\sigma) = \mathrm{Ver1}^O(\mathrm{Ver0}^O(\mathsf{vrk},h),\sigma)$ as follows: $\mathrm{Ver0}^O(\mathsf{vrk},h)$ generates a random ciphertext c for a random message m relative to identity h as $c \leftarrow \mathsf{Enc}^{\mathbf{e}}(\mathsf{pp},h,m)$, lets QEnc contain the Q-A pairs (which are only of $\mathbf{e}$-type), and outputs (m,c).[10] Then, $\mathrm{Ver1}^O$, given $\sigma := (\mathsf{sk}_h,\mathsf{QGen}_h)$ and (m,c), simply decrypts $\mathsf{Dec}^{\mathbf{d}}(\mathsf{sk}_h,c)$ and outputs 1 iff the decryption outputs m. The signature correct, and is also secure: if an adversary can forge a signature on a target message h, it can also decrypt ciphertexts for that target index h under the PKCom scheme. (Under the PKCom scheme, the adversary can have the secret keys for all but the target index, since the public keys for all non-target indices are submitted by the adversary itself. Thus, a PKCom adversary can sample vrk itself.) The signature, however, is not restricted because $\mathrm{Ver1}^O$ makes queries in order to decrypt.

Making the signature restricted. A first (naive) idea in making Ver1 oracle-free is to have $\mathrm{Ver0}^O$ pass QEnc, in addition to (vrk,m,c), onto Ver1, and let $\mathrm{Ver1}((\mathsf{vrk},m,c,\mathsf{QEnc}),\sigma)$, where $\sigma := (\mathsf{sk}_h,\mathsf{QGen}_h)$, decrypt $\mathsf{Dec}^{\mathbf{d}}(\mathsf{sk}_h,c)$ while using QEnc and $\{\mathsf{QGen}_i\}$ as hints, and respond to queries whose answers cannot be determined based on these hints with random values. In more detail, for

[9] The Q-A sets QGen_i's will not be used in this simple construction, but later one they will be used when we make the signature restricted.

[10] Again, the set QEnc will not be used in this (flawed) construction, but will be used later when we discuss the fixes.

an emerged query (which can only be of $\mathbf{d}$-type) $\mathsf{qu} := ((\mathsf{tk}, y) \underset{\mathbf{d}}{\rightarrow} ?)$ during $\mathsf{Dec}^{\mathbf{d}}(\mathsf{sk}_h, c)$ if both of the following hold then respond to qu with x:

(a) There exists a Q-A pair $(\mathsf{tk} \underset{\mathbf{g}}{\rightarrow} \mathsf{ik}) \in \cup_i \mathsf{QGen}_i$ for some ik; and

(b) there exists $((\mathsf{ik}, x) \underset{\mathbf{e}}{\rightarrow} y) \in \mathsf{QEnc}$ for some x.

Otherwise (i.e., if at least one of the above does not hold), pick a random answer. We claim we have correctness. This is because $(\mathsf{pk}_i, \mathsf{sk}_i)$ pairs are all generated honestly by running $\mathsf{Key}^{\mathbf{g}}$, with $\{\mathsf{QGen}_i\}$ being their underlying Q-A pairs, and that all of $\mathbf{d}$ queries during Dec are responded to consistently with those of $\{\mathsf{QGen}_i\}$ and QEnc.

However, we cannot reduce the security of the signature to that of the original PKCom (so to argue security). For example, suppose the PKCom's user public-secret key pairs $(\mathsf{pk}_i, \mathsf{sk}_i)$ are simply random index/trapdoor key pairs $(\mathsf{ik}_i, \mathsf{tk}_i)$ generated by calling $\mathbf{g}$ (i.e., $\mathsf{pk}_i = \mathsf{ik}_i$). An adversary $\mathcal{A}$ may then forge a signature as $\sigma' := (\widetilde{\mathsf{tk}}_h, (\widetilde{\mathsf{tk}}_h \underset{\mathbf{g}}{\rightarrow} \mathsf{ik}_h))$, where $\widetilde{\mathsf{tk}}_h$ is just a 'junk' value. By Conditions (A) and (b) above, $\mathrm{Ver1}((\mathsf{vrk}, m, c, \mathsf{QEnc}), \sigma')$ will always decrypt c to m, outputting 1. But the forger $\mathcal{A}$ has not done anything clever, so we cannot use it in a reduction to break the security of $\mathsf{PKCom}^{\mathbf{g},\mathbf{e},\mathbf{d}}$. The reason we cannot prove a security reduction (for arguing the signature is as secure as the base PKCom) is that the verification provides 'free lunch' to a forger: inverting images with respect to an ik_h whose trapdoor key may not be known to the forger.

Compression to the rescue. So far, we have not used the fact that pp is compact (of size $\ll n$), so it is not a surprise we cannot establish a security reduction from the signature to the PKCom. In other words, by plugging in a trivial PKCom scheme whose pp contains all public keys, we get a restricted signature against which there exists a generic attack, but the base PKCom is secure! We should use the fact that $|\mathsf{pp}|$ is compact in order to avoid giving free lunch to a forger in the above sense. Call an ik valid if $\mathsf{ik} \in \mathbf{g}(*)$. Assume $\mathbf{g}$ is sufficiently length increasing so the only way to produce a valid ik is to call $\mathbf{g}$ on a preimage. Our main idea is as follows: letting QGen_i be as above (all formed honestly), there must exist an index $h \in [n]$ such that the set of all valid ik's that emerge as $((\mathsf{ik}, *) \underset{\mathbf{e}}{\rightarrow} ?)$ queries during a random PKCom encryption $\mathsf{Enc}^{\mathbf{e}}(\mathsf{pp}, h, *)$ to index h are a subset of those ik's for which we have $(* \underset{\mathbf{g}}{\rightarrow} \mathsf{ik}) \in \cup_{i \neq h} \mathsf{QGen}_i$. Call this Condition cover. We will show in a moment why this condition holds, but let us sketch how to modify $\mathrm{Ver}^{O}(\mathsf{vrk}, h, \sigma) = \mathrm{Ver1}(\mathrm{Ver0}^{O}(\mathsf{vrk}, h), \sigma)$ in light of this fact. First, $\mathrm{Ver0}^{O}(\mathsf{vrk}, h)$ outputs (m, c, QEnc), as before. Now $\mathrm{Ver1}((\mathsf{vrk}, m, c, \mathsf{QEnc}), \sigma)$, where $\sigma := (\mathsf{sk}_h, \mathsf{QGen}_h)$, proceeds exactly as before, except in response to a query $\mathsf{qu} := ((\mathsf{tk}, y) \underset{\mathbf{d}}{\rightarrow} ?)$, Condition (a) above will change to

(A) there exists a Q-A pair $(\mathsf{tk} \underset{\mathbf{g}}{\rightarrow} \mathsf{ik}) \in \cup_{i \neq h} \mathsf{QGen}_i$ for some ik.

Now correctness still holds, thanks to Condition cover. (Any $((\mathsf{ik}, x) \underset{\mathbf{e}}{\rightarrow} y)$ for a valid ik has a matching trapdoor key $(\mathsf{tk} \underset{\mathbf{g}}{\rightarrow} \mathsf{ik}) \in \cup_{i \neq h} \mathsf{QGen}_i$. All other decryp-

tion queries may be responded to randomly, without creating inconsistencies.) We should now have security (at least intuitively), because we do not provide free lunch to a forger anymore: Ver1 inverts images only for ik's already covered in $(\mathsf{tk} \underset{\mathbf{g}}{\rightarrow} \mathsf{ik}) \in \cup_{i \neq h} \mathsf{QGen}_i$, but this information is already available to the adversary itself (as part of vrk). Making this intuition formal, however, requires some delicate reasoning, which we skip here.

Proving condition cover *via a compression technique.* Recall that $\mathrm{Enc}^{\mathbf{e}}(\mathsf{pp}, *, *)$ only calls $\mathbf{e}$, and the only information it gets regarding $\mathbf{g}$ is via pp which in turn has size $\ll n$. It is not hard to see if Condition cover does not hold, then by performing random encryptions for all indices $\mathrm{Enc}^{\mathbf{e}}(\mathsf{pp}, 1, *), \ldots, \mathrm{Enc}^{\mathbf{e}}(\mathsf{pp}, n, *)$, we will end up calling $\mathbf{e}(\mathsf{ik}, *)$ upon at least n different valid ik's. To see this let Q_i contain any ik such that $(* \underset{\mathbf{g}}{\rightarrow} \mathsf{ik}) \in \mathsf{QGen}_i$. If $\overline{\mathsf{cover}}$ holds, during $\mathrm{Enc}^{\mathbf{e}}(\mathsf{pp}, u, *)$, for any $u \in [n]$, we will make a query $((\mathsf{ik}_u, *) \underset{\mathbf{e}}{\rightarrow} ?)$ for a valid ik_u where $\mathsf{ik}_u \notin \cup_{i \neq u} \mathsf{QGen}_i$. We claim all ik_i are distinct, so we will have n distinct valid id_i's. Assume the contrary and that $\mathsf{ik}_1 = \mathsf{ik}_2$. We know both $\mathsf{ik}_1, \mathsf{ik}_2 \in \cup_i \mathsf{Q}_i$, because a valid ik cannot come out of thin air. We also know $\mathsf{ik}_1 \notin \cup_{i \neq 1} Q_i$, and so $\mathsf{ik}_1 \in \mathsf{Q}_1 \setminus \cup_{i \neq 1} \mathsf{Q}_i$. So, if $\mathsf{ik}_1 = \mathsf{ik}_2$, then $\mathsf{ik}_2 \in \mathsf{Q}_1 \setminus \cup_{i \neq 1} \mathsf{Q}_i$, but this contradicts the fact that $\mathsf{ik}_2 \notin \cup_{i \neq 2} \mathsf{Q}_i$.

Theorem 2 (Compression, informal). *Let* pp *be any advice string of size* $\ll n$ *generated based on* $(\mathbf{g}, \mathbf{e}, \mathbf{d})$*. For any poly-query adversary* $\mathcal{A}^{\mathbf{g},\mathbf{e},\mathbf{d}}(\mathsf{pp})$*, the probability that* $\mathcal{A}$ *can output a list* L *of* ik*'s satisfying the following two conditions is negligible: (a)* L *has at least* n *distinct valid* ik*'s, and (b) no* ik *in* L *was generated as a result of a* $\mathbf{g}$ *query by* $\mathcal{A}$ *itself.*

We show if there exists an adversary in the sense of the above theorem, then one can compress the oracle $\mathbf{g}$. We prove this via Gennaro-Trevisan style compression techniques [GT00], later generalized by Haitner *et al.* [HHRS07]. In our theorem description, the adversary $\mathcal{A}$ does not need to know which of the n ik's are valid: as long as at least n of them are valid we will prove a compression. Our theorem description holds even if the adversary makes an exponential number of queries (for a carefully selected exponential). We believe this technique employed within black-box separations might find other applications.

Allowing a CRS. We will now sketch how things will be different, still in the limited-access setting $(\mathsf{Key}^{\mathbf{g}}, \mathsf{Com}^{\mathbf{g}}, \mathsf{Enc}^{\mathbf{e}}, \mathsf{Dec}^{\mathbf{d}})$, by allowing in a CRS. Suppose the CRS is generated as $\mathsf{crs} \leftarrow \mathsf{CRS}(1^\kappa, 1^n)$, and is used in key generation $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Key}^g(\mathsf{crs})$. Let us first explain what will go wrong if we leave the above approach unmodified. Recall that $\mathrm{Ver1}((m, c, \mathsf{QEnc}), \sigma)$, where $\sigma := (\mathsf{sk}_h, \mathsf{QGen}_h)$, decrypts $\mathsf{Dec}^{\mathbf{d}}(\mathsf{sk}_h, c)$ and handles a query $\mathsf{qu}\colon ((\mathsf{tk}, c) \underset{\mathbf{d}}{\rightarrow} ?)$ via Conditions (A) and (b) above. We were able to argue correctness because for some index h with all but negligible probability, all valid ik's upon which we have a query $((\mathsf{ik}, *) \underset{\mathbf{e}}{\rightarrow} ?)$ must have been collected in $\cup_i \mathsf{QGen}_{i \neq h}$. However, this fact breaks down in the CRS case, because $\mathrm{Enc}^{\mathbf{e}}(\mathsf{pp}, h, *)$ might call $\mathbf{e}$ upon an ik

that comes from crs, and whose trapdoor key is not collected in any of $\cup_i \mathsf{QGen}_i$. Thus, responding to inversion queries relative to $\mathbf{e}(\mathsf{ik}, *)$ with random values during decryption will create an inconsistency, destroying correctness. Since we aim to argue correctness, suppose $(\mathsf{sk}_h, \mathsf{QGen}_h)$ were generated honestly. Our solution is based on the following intuition: If a query $((\mathsf{tk}, *) \underset{\mathbf{d}}{\rightarrow} ?)$, where $\mathbf{g}(\mathsf{tk}) = \mathsf{ik}$, emerges during decryption of a random encryption relative to $\mathrm{Enc}^{\mathbf{e}}(\mathsf{pp}, h, *)$ with good probability, then by choosing many $(\mathsf{sk}_h, \mathsf{QGen}_h)$ and forming random encryptions as $\mathrm{Enc}^{\mathbf{e}}(\mathsf{pp}, h, *)$ and decrypting them back (all using the real oracles) we should collect these tk values. This encrypt-decrypt process will be performed by $\mathrm{Gen}^{O}(1^n, h \in [n])$ (the key generation algorithm of the signature scheme) and all the Q-A pairs are appended to a list T, put in the final vrk. Back to the above discussion, $\mathrm{Ver1}((m, c, \mathsf{QEnc}), \sigma)$ will decrypt $\mathsf{Dec}^{\mathbf{d}}(\mathsf{sk}_h, c)$ as per QEnc if (I) and (b) holds, where (b) is as above and (I) is as follows.

(I) There exists a Q-A pair $(\mathsf{tk} \underset{\mathbf{g}}{\rightarrow} \mathsf{ik}) \in \mathsf{T} \cup_{i \neq h} \mathsf{QGen}_i$ for some ik.

The proof of security is similar to the previous case, based on the intuition illustrated before.

Finally, for the general case in which oracle access is unrestricted (e.g., $\mathsf{Key}^{\mathbf{g},\mathbf{e},\mathbf{d}}$) we define the notion of 'free lunch' (which might make room for forgeries) as inversions $\mathbf{e}^{-1}(\mathsf{ik}, *)$, where $(* \underset{\mathbf{g}}{\rightarrow} \mathsf{ik})$ never appears in $\mathsf{QGen}_{i \neq h}$, nor in any safe lists (e.g., T as explained above, or any ik such that $(* \underset{\mathbf{g}}{\rightarrow} \mathsf{ik})$ is generated during $\mathrm{Enc}^{\mathbf{g},\mathbf{e},\mathbf{d}}(\mathsf{pp}, h, *)$). The chief challenge is to strike a delicate balance during the decryption performed by $\mathrm{Ver1}((m, c, \mathsf{QEnc}), \sigma)$ in between not overtly answering all $\mathbf{d}(\mathsf{tk}, y)$ queries as per QEnc (which will violate security) and not answering any queries at all (which will destroy correctness). The main techniques for establishing such a balance were sketched above, but the whole analysis for the general case requires more involved reasoning.

Impossibility in Shoup's GGM. We now describe how to derive a restricted signature scheme $\mathcal{E}^{\mathbb{G}_{RR}}$ from a PKCom construction $\mathsf{PKCom}^{\mathbb{G}_{RR}}$, where the oracle $\mathbb{G}_{RR} := (\mathbf{label}, \mathbf{add})$ comes with a labeling oracle $\mathbf{label}$ producing labels for exponents in $\mathbb{Z}_p$ (where p is the order of the group) and $\mathbf{add}$ adds two labels. We assume $\mathbf{label}$ produces labels of sufficiently large length, so that producing a label without calling $\mathbf{label}$ or $\mathbf{add}$ is effectively impossible. The general methodology is as follows, but we need some additional ideas to deal with the algebraic structure imposed bu groups. To illustrate our core ideas suppose we have no crs and that Com makes no $\mathbb{G}_{RR}$ queries, so the PKCom construction is as $(\mathsf{Key}^{\mathbb{G}}_{RR}, \mathsf{Com}, \mathsf{Enc}^{\mathbb{G}}_{RR}, \mathsf{Dec}^{\mathbb{G}}_{RR})$. Assume wlog all algorithms only have access to $\mathbf{add}$ (access to $\mathbf{label}$ can be simulated by including the generator as part of every input). The algorithms $\mathrm{Gen}^{\mathbb{G}}_{RR}$ and $\mathrm{Sig}^{\mathbb{G}}_{RR}$ and $\mathrm{Ver0}^{\mathbb{G}_{RR}}(\mathsf{vrk}, h)$ are defined exactly as before. A naive idea for $\mathrm{Ver1}(\mathsf{vrk}, m, \sigma)$, where $\sigma := ((\mathsf{sk}_h, \mathsf{QGen}_h))$, is to answer to an $\mathbf{add}$ query $((\ell_1, \ell_2) \underset{\mathbf{add}}{\longrightarrow} ?)$ according to what can be inferred from the collective set of Q-A pairs in $\cup_i \mathsf{QGen}_i \cup \mathsf{QEnc}$. Namely, consider a matrix M with columns labeled according to labels present in output responses to

queries in $\cup_i \mathsf{QGen}_i \cup \mathsf{QEnc}$. A given Q-A pair $((\ell, \ell') \xrightarrow[\mathbf{add}]{} \ell'')$ in $\cup_i \mathsf{QGen}_i \cup \mathsf{QEnc}$ is embedded into the matrix M by adding a row, which has a 1 on the ℓ-labelled column, a 1 on the ℓ' labeled column and a -1 on the ℓ''-labeled column. For brevity, we denote such a row with $x_\ell + x_{\ell'} - x_{\ell''}$. Having formed this matrix M, suppose Ver1 given a query $((\ell_1, \ell_2) \xrightarrow[\mathbf{add}]{} ?)$ checks if an answer is present in M: namely, if there exists a label ℓ^* such that $x_{\ell_1} + x_{\ell_2} - x_{\ell^*} \in \mathsf{Span}(M)$, where Span denotes row span; if so, respond with ℓ^*, otherwise with a random label. This restricted signature scheme is correct (similarly to how we argued before), but is not secure in the sense of having a security reduction from the signature to the base PKCom. We will now establish a balancing act (in the sense of before) as follows. Call a label Known if $\mathbf{label}^{-1}(\ell)$ (its discrete log) is recoverable given $\cup_{i \neq h} \mathsf{QGen}_i \cup \mathsf{QEnc}$. For GGMs, we will prove a compression theorem (stated later), which as a consequence implies the following $\mathsf{covering}$ condition: with all but negligible probability, there exists an index h such that during a random encryption $\mathsf{Enc}^{\mathbf{add}}(\mathsf{pp}, h, *)$, if there exists a Q-A pair $((\ell_1, \ell_2) \xrightarrow[\mathbf{add}]{} \ell^*)$ such that $\ell^* \neq \perp$, then $\ell_1, \ell_2 \in \mathsf{Known}$. That is, for $b \in \{0, 1\}$, ℓ_b is either already a label in $\cup_{i \neq h} \mathsf{QGen}_i$, or is obtained via a sequence of **add** operations on labels with known discrete logs. With this intuition mind, Ver1 simulates the response to an **add** query $((\ell_1, \ell_2) \xrightarrow[\mathbf{add}]{} ?)$ as follows: if there exists a Known label ℓ^* such that such that $x_{\ell_1} + x_{\ell_2} - x_{\ell^*} \in \mathsf{Span}(M)$, where Span denotes row span; if so, respond with ℓ^*, else with a random label. This relaxation enables us to prove that no security is lost in the process. Finally, we derive the $\mathsf{covering}$ condition above from a compression lemma that we develop and prove in GGMs, which states given a short advice string pp, one can extract a bounded number of valid ℓ labels (those in the output of **label**).

Theorem 3 (Informal). *Let* pp *be any advice string of size* $\ll n$ *generated based on* $(\mathbf{label}, \mathbf{add})$*. For any poly-query adversary* $\mathcal{A}^{\mathbf{label}, \mathbf{add}}$*, the probability that* $\mathcal{A}$ *can output a list* L *of* ℓ*'s satisfying the following two conditions is negligible: (a)* L *has at least* n *distinct valid* ℓ*'s, and (b) no* ℓ *in the list was generated as a result of a* **label** *or a* **add** *query.*

All the statements mentioned after Theorem 2 also hold for Theorem 3. The proof of this is based on a generalization of the Gennaro-Trevisan compression techniques [GT00] to the GGM setting. The GGM setting, due to its algebraic nature, makes the compression argument more delicate and challenging. Our techniques may be of independent interest.

Why going through restricted signature instead of a direct proof. Even though we could write a direct proof that avoids going through the notion of restricted signatures, by using this approach we also get the benefits of the proof of [Zha22] for free. In particular, a direct proof would involve the high-level approach of (1) learning useful information about the oracle, (2) doing a reverse sampling of the keys and faking a partial oracle to be used for decrypting the challenge, and (3) analyzing the attack through careful hybrids. However, using the restricted

signature approach allows us to have a more modular proof. In particular, with this approach Step (2) above would not be needed; all we do is a black-box reduction between RBE and restricted signatures, and the attack on RBE follows from the poly-query attack on the signatures that is transformed into an attack on RBE through the black-box reduction between them. In addition to simplicity and modularity, as a bonus we could also better explain (in the next paragraph) the new challenges that arise for our separation proof for RBE, in comparison with IBE, and how we resolve them.

3 Preliminaries

Notation. We use κ for the security parameter. We write $\mathrm{negl}(x)$ for a negligible function of x. For an integer n, $[n] := \{1, \ldots, n\}$. We write $x \leftarrow \mathcal{S}$ (resp., $x \leftarrow D$) to denote picking an element uniformly at random from a set $\mathcal{S}$ (resp., a distribution D). By $\Pr[E; D]$ we denote the probability of E where D is random space/process over which E is defined. In general, we use $*$ as a placeholder for an arbitrary value. For example, $(x, *, *)$ is a 3-tuple where the first value is x and the second and third values are two arbitrary values. For function f, then $f(*)$ could denote the range of f (we let the context clarify whether $f(*)$ refers to a set or a single unspecified value). For an oracle $O = (\mathbf{q}_1, \ldots, \mathbf{q}_n)$ consisting of n different types of queries, we use $(x \underset{\mathbf{q}_i}{\longrightarrow} y)$ to denote one query of type $\mathbf{q}_i$ where the input is x and the output is y. Note that both x and y can be tuples. Also note that we do not use $*$ as a placeholder for y or any term of y since y is not arbitrary but depends on x and $\mathbf{q}_i$. Finally, we explain how we use $\subset$ in this paper. (Note that $\subseteq$ is used in a similar way.) Since a function is formally defined as a relation, which is a set, when we write $f \subset g$ for two functions f, g, we mean f (viewed as a relation) is a subset of g (also viewed as a relation), which means that the domain of f is a subset of the domain of g and it is defined similarly on those points. For two n-tuples $x = (x_1, \cdots, x_n)$ and $y = (y_1, \cdots, y_n)$, $x \subset y$ if and only $x_i \subset y_i$ for every $i \in [n]$. Since an oracle can be viewed as an n-tuple of functions, the notation $O' \subset O$ is well defined for two oracles O, O'.

3.1 Public Key Compression

Here we define Public Key Compression (PKCom). This primitive allows n identities $\mathsf{id}_1, \ldots, \mathsf{id}_n$ to independently sample their public/secret key pairs $(\mathsf{pk}_1, \mathsf{sk}_1), \ldots, (\mathsf{pk}_n, \mathsf{sk}_n)$ and then broadcast $(\mathsf{id}_1, \mathsf{pk}_1), \ldots, (\mathsf{id}_n, \mathsf{pk}_n)$. There is then a compression algorithm Com that compresses all these public keys into a short public parameter pp. This pp together with id_i can be used to encrypt to id_i. Finally, user id_i can use her secret key sk_i and the set of all public keys $\mathsf{pk}_1, \ldots, \mathsf{pk}_n$ to decrypt any message encrypted to her. In the actual definition of RBE [GHMR18], of which PKCom is a special case [HLWW23], a user needs only a short public "decryption-update" string (which in turn is deterministically derived from $\mathsf{pk}_1, \ldots, \mathsf{pk}_n$) to be able to perform decryption. But since we aim to prove a lowerbound, by allowing the decryption algorithm to take in all of

$\mathsf{pk}_1, \ldots, \mathsf{pk}_n$, our impossibility results will become only stronger. Also, we assume the n identities are $\{\mathsf{id}_1 = 1, \ldots, \mathsf{id}_n = n\}$ for simplicity, and state the scheme for a *key encapsulation* variant; again, both of these will only make our impossibility results stronger.

Definition 1. *A* public key compression *scheme consists of PPT algorithms* $(\mathsf{CRS}, \mathsf{Key}, \mathsf{Com}, \mathsf{Enc}, \mathsf{Dec})$*:*

- ***CRS generation.*** $\mathsf{CRS}(1^\kappa, 1^n) \to \mathsf{crs}$ *is a randomized algorithm that takes in a security parameter* κ *and an integer* n *(the number of parties), and outputs a CRS* crs *of length* $\mathsf{poly}(\kappa, n)$*. We allow* crs *to grow polynomially with the number of users.*
- ***Key generation.*** $\mathsf{Key}(1^\kappa, \mathsf{crs}) \to (\mathsf{pk}, \mathsf{sk})$ *takes in* 1^κ *and* crs *and outputs a pair of public and secret keys* $(\mathsf{pk}, \mathsf{sk})$*.*
- ***Key compression.*** $\mathsf{Com}(\mathsf{crs}, \{\mathsf{pk}_i\}_{i\in[n]}) \to \mathsf{pp}$ *takes in the security parameter, the* crs*, a list of public keys* $\{\mathsf{pk}_i\}_{i\in[n]}$*, and deterministically outputs* pp *as the compressed public key.*
- ***Encryption.*** $\mathsf{Enc}(\mathsf{pp}, \mathsf{id}) \to (\mathsf{m}, \mathsf{ct})$ *takes in* pp*, a recipient identity* $\mathsf{id} \in [n]$*, and outputs a random* $\mathsf{m} \leftarrow \{0,1\}^\kappa$ *and a corresponding ciphertext* ct*.*
- ***Decryption.*** $\mathsf{Dec}(\mathsf{crs}, \mathsf{id}, \mathsf{sk}, \{\mathsf{pk}_i\}_{i\in[n]}, \mathsf{ct}) \to \mathsf{m}$ *takes in* crs*, an identity* $\mathsf{id} \in [n]$*, a secret key* sk*, public keys* $\{\mathsf{pk}_i\}_{i\in[n]}$*, a ciphertext* ct*, and outputs a plaintext* m *or a special symbol* $\perp$*.*

We require completeness, compactness and security, as defined next.

- ***Completeness:*** *The decryption algorithm recovers the plaintext with all but negligible probability. For every* $n \in \mathbb{N}$*, any* $i \in [n]$*,* $\mathsf{crs} \leftarrow \mathsf{CRS}(1^\kappa, n)$*,* $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{Key}(1^\kappa, \mathsf{crs})$*,* $(m, \mathsf{ct}) \leftarrow \mathsf{Enc}(\mathsf{pp}, \mathsf{id})$*, it holds that* $\Pr[\mathsf{Dec}(\mathsf{crs}, \mathsf{id}, \mathsf{sk}_i, \{\mathsf{pk}_i\}_{i\in[n]}, \mathsf{ct}) = m] \geq 1 - \mathrm{negl}(\kappa)$*.*
- ***Compactness:*** *There exists a fixed polynomial* poly *such that for all* n *and* pp *formed as above,* $|\mathsf{pp}| = o(n)\mathsf{poly}(\kappa)$*. We require sub-linear compactness, making our impossibility results stronger.*
- ***Security:*** *Any PPT adversary* $\mathcal{A}$ *has a negligible advantage in the following game.* $\mathcal{A}$ *is given* n *and a CRS* $\mathsf{crs} \leftarrow \mathsf{CRS}(1^\kappa, n)$*, and* $\mathcal{A}$ *outputs a challenge index* h *and* $n-1$ *public keys* $\{\mathsf{pk}_i\}_{i\neq h}$*. The challenger samples* $(\mathsf{pk}_h, *) \leftarrow \mathsf{Key}(1^\kappa, \mathsf{crs})$*, forms* $\mathsf{pp} := \mathsf{Com}(\mathsf{crs}, \{\mathsf{pk}_i\}_{i\in[n]})$*, and* $(m, \mathsf{ct}) \leftarrow \mathsf{Enc}(\mathsf{pp}, \mathsf{id})$*.* $\mathcal{A}$ *is given* ct*, and outputs* m' *and wins if* $m' = m$*.*

Note that we are making the security notion weaker (the adversary's job is more difficult); our impossibility results separates this weak notion of security, hence making our results stronger.

4 Impossibility of PKCom from TDPs

In this section, we show that there exists an oracle $\mathcal{O}$ relative to which TDPs exists but PKCom does not. We define a distribution on TDP oracles as follows.

Definition 2. *We define an oracle distribution Ψ whose samples are oracles of the form $\mathbf{O} = (\mathbf{g}, \mathbf{e}, \mathbf{d})$. The distribution is parameterized over a security parameter κ, but we keep it implicit for better readability.*

- $\mathbf{g}\colon \{0,1\}^{\kappa} \mapsto \{0,1\}^{3\kappa}$ *is a random injective length-tripling function, mapping a trapdoor key to an index key.*
- $\mathbf{e}\colon \{0,1\}^{3\kappa} \times \{0,1\}^{\kappa} \mapsto \{0,1\}^{\kappa}$ *is a random function under the following condition: for all $\mathsf{ik} \in \{0,1\}^{3\kappa}$, the function $\mathbf{e}(\mathsf{ik}, \cdot)$ is a permutation.*
- $\mathbf{d} : \{0,1\}^{\kappa} \times \{0,1\}^{\kappa} \mapsto \{0,1\}^{\kappa}$ *is the inversion oracle, where $\mathbf{d}(\mathsf{tk}, y)$ outputs $x \in \{0,1\}^{\kappa}$ iff $\mathbf{e}(\mathbf{g}(\mathsf{tk}), x) = y$.*

Definition 3 (Validity of partial oracles). *We say a partial oracle $\mathbf{O}'$ (defined only on a subset of all points) is Ψ-valid if for some $\mathbf{O} \in \mathsf{Supp}(\Psi) : \mathbf{O}' \subseteq \mathbf{O}$, where Supp denotes the support of a distribution. We say an oracle $(\mathbf{g}, \mathbf{e}, \mathbf{d})$ is TDP-valid if it satisfies TDP's perfect completeness. A partial TDP-valid oracle is one which is a subset of a TDP-valid oracle (i.e., a triple (g, e, d) that satisfies TDP correctness, but which may not be in the support of Ψ). Note that any Ψ-valid oracle is TDP-valid as well. We say a partial oracle $\mathbf{O}'$ is TDP-consistent with a set of Query/Answer (Q-A in short) pairs S if $\mathbf{O}' \cup \mathsf{S}$ is TDP-valid.*

4.1 Oracle-Based Target-Restricted Signatures

Toward proving our impossibility results, inspired by [Zha22], we define the notion of oracle-aided target-restricted signatures. The signature's message space is $[n]$, and we require correctness and security to hold with respect to a single, random target point, based on which signing and verification keys are generated. We first present the definition and then compare it to that of [Zha22].

Definition 4 (Target-restricted signatures [Zha22]). *Let $n = \mathsf{poly}(\kappa)$. An n-target restricted signature scheme $(\mathrm{Gen}^O, \mathrm{Sig}^O, \mathrm{Ver}^O)$ relative to an oracle O is defined as follows. $\mathrm{Gen}^O(1^{\kappa}, m) \to (\mathsf{sgk}, \mathsf{vrk})$: takes in a security parameter and a target message $m \in [n]$, and outputs a signing key sgk and a verification key vrk. The other algorithms are defined as in standard signature schemes. We require the following properties.*

- *δ-**target correctness**:*

$$\Pr[\mathrm{Ver}^O(\mathsf{vrk}, m, \mathrm{Sig}^O(\mathsf{sgk}, m)) = 1; m \leftarrow [n], (\mathsf{sgk}, \mathsf{vrk}) \leftarrow \mathrm{Gen}^O(1^{\kappa}, m)] \geq \delta,$$

  where the probability is taken over $m \leftarrow [n]$, $(\mathsf{sgk}, \mathsf{vrk}) \leftarrow \mathrm{Gen}^O(1^{\kappa}, m)$ and the random coins used by Sig^O and Ver^O.
- ***Restricted structure**: We have $\mathrm{Ver}^O(\mathsf{vrk}, m, \sigma) = \mathrm{Ver1}(\mathrm{Ver0}^O(\mathsf{vrk}, m), \sigma)$, where $\mathrm{Ver1}$ makes no oracle calls.*
- ***Zero-time unforgeability**: For any PPT adversary $\mathcal{A}$,*

$$\Pr[\mathrm{Ver}^O(\mathsf{vrk}, m, \sigma) = 1; m \leftarrow [n], (\mathsf{sgk}, \mathsf{vrk}) \leftarrow \mathrm{Gen}^O(1^{\kappa}, m), \sigma \leftarrow \mathcal{A}^O(\mathsf{vrk}, m)] \leq \mathrm{negl}(\kappa).$$

Zhandry [Zha22] defined oracle-based restricted signatures, where signing and verification keys should work for all messages, and proved such signatures are impossible relative to any oracle. Namely, there exists an adversary that can forge a signature by making at most a polynomial number of queries to the oracle, and by performing possibly exponential-time computations independent of the oracle. In the setting of [Zha22] the message space is of exponential size, but in our setting the message space is $[n]$ and the verification key is allowed to grow with $[n]$. These differences are useful for our setting as we will derive the existence of *target*-restricted signatures during our impossibility proofs. Despite these differences, the following lemma shows that Zhandry's proof, that restricted signatures do not exist, extends almost immediately to our target-restricted setting.

Lemma 1 (Adapted from Lemma 7.4 in [Zha22]). *Let $1 \geq \delta > 0$ and O be an oracle. For any target-restricted signature Λ relative to O that has δ target correctness according to Definition 4, there exists a computationally unbounded adversary which makes only polynomially many queries to O that breaks Λ with advantage at least $\delta^3/100$.*

The proof of the above lemma is basically the same as the proof of Lemma 7.4 in [Zha22]. At a high level, the proof crucially relies on the restricted structure of the verification algorithm. The key idea of the proof is that since $\mathrm{Ver}^O(\mathsf{vrk}, m, \sigma) = \mathrm{Ver1}(\mathrm{Ver0}^O(\mathsf{vrk}, \mathsf{m}), \sigma)$, a computationally unbounded adversary can first compute an intermediate value $v = \mathrm{Ver0}^O(\mathsf{vrk}, \mathsf{m})$ by itself and then brute force search over the circuit $\mathrm{Ver1}(v, \cdot)$ for a valid signature σ satisfying $\mathrm{Ver1}(v, \sigma) = 1$. Since target-restricted signatures also have the same restricted structure of verification algorithm, the same proof works. For sake of completeness, in Appendix A we include a full of Lemma 1, which is heavily based on that of [Zha22] and is simply adapted to our setting.

Equipped with Lemma 1, we show any TDP-oracle-based PKCom may be transformed into an oracle-based target-restricted signatures, hence obtaining an impossibility result. As a warm-up and to show our core techniques, we first present this transformation for the CRS-free case in Section 4.2 and then present the transformation for schemes with CRS in Section B.1.

4.2 Impossibility of CRS-free PKCom from TDP

We first present the transformation to target-restricted signatures for the case in which the PKCom does not have a CRS. Recall the notions of correctness and security of PKCom given in Definition 1. These notions are defined analogously relative to any fixed oracle $O = (\mathbf{g}, \mathbf{e}, \mathbf{d})$.

Theorem 4. *For $\epsilon := \frac{1}{\mathsf{poly}(\kappa)}$ let $\mathcal{E}^{\mathbf{g},\mathbf{e},\mathbf{d}} := (\mathsf{Key}^{\mathbf{g},\mathbf{e},\mathbf{d}}, \mathsf{Com}^{\mathbf{g},\mathbf{e},\mathbf{d}}, \mathsf{Enc}^{\mathbf{g},\mathbf{e},\mathbf{d}}, \mathsf{Dec}^{\mathbf{g},\mathbf{e},\mathbf{d}})$ be a $(1-\epsilon)$-correct PKCom scheme with respect to a random TDP oracle $O = (\mathbf{g}, \mathbf{e}, \mathbf{d})$. Suppose a public parameter pp under $\mathcal{E}^{\mathbf{g},\mathbf{e},\mathbf{d}}$ satisfies $|\mathsf{pp}| \leq \frac{(n-2)|\mathsf{ik}|}{2}$, where n is the number of users and ik is a base index key (recall $|\mathsf{ik}| = 3\kappa$, Definition 2). Then, there exists a $(1-\epsilon)\frac{(1-2^{-\kappa/3})}{n}$-correct target-restricted signature scheme relative to $O = (\mathbf{g}, \mathbf{e}, \mathbf{d})$.*

Note 1. For all oracle algorithms $A^{\mathbf{g},\mathbf{e},\mathbf{d}}$ considered throughout, we assume whenever a Q-A $((\mathsf{tk}, y) \underset{\mathbf{d}}{\rightarrow} x)$ is made by $A^{\mathbf{g},\mathbf{e},\mathbf{d}}$, two dummy queries $(\mathsf{tk} \underset{\mathbf{g}}{\rightarrow} ?)$ and $((\mathsf{ik}, x) \underset{\mathbf{e}}{\rightarrow} ?)$ are subsequently made, where $\mathsf{ik} = \mathbf{g}(\mathsf{tk})$. Thus, whenever $((\mathsf{tk}, y) \underset{\mathbf{d}}{\rightarrow} x)$ is in A's Q-A list, so are $(\mathsf{tk} \underset{\mathbf{g}}{\rightarrow} \mathsf{ik})$ and $((\mathsf{ik}, x) \underset{\mathbf{e}}{\rightarrow} y)$. Moreover, for any A as above, we assume whenever two Q-A pairs $(\mathsf{tk} \underset{\mathbf{g}}{\rightarrow} *)$ and $((\mathsf{ik}, x) \underset{\mathbf{e}}{\rightarrow} y)$ are made first, then no subsequent query $((\mathsf{tk}, y) \underset{\mathbf{d}}{\rightarrow} ?)$ is ever made.

Bluebird view of the proof of Theorem 4. We show how to transform PKComs into target restricted signatures. This is given in Construction 5. The construction is similar to Lamport's signatures from OWFs, adapted to the PKE setting. That is, we generate n public keys $\{\mathsf{pk}_i\}$, put all public keys and all secret keys except the target (hth) one in the verification key. The signature σ on $h \in [n]$ is a secret key for pk_h. The verification function will encrypt a random message m relative to h (performed inside Ver0), and decrypts the ciphertext c inside Ver1 using a signature $\sigma := \mathsf{sk}_h$ to see if it gets m back. First, it is clear that Ver0 can be performed before seeing σ. The most major step is to make sure Ver1 can decrypt c without making queries. To this end, we equip the verification key with Q-A pairs underlying $\{\mathsf{pk}_i\}_{i \neq h}$ (called QGen_i in the construction), and we also let Ver0 pass on all Q-A pairs QEnc underlying c to Ver1. The algorithm Ver1 simulates responses to its queries using these sets. The main difficulty is to define the simulated decryption in such a way that we can establish both correctness and security. (For example, letting Ver1 invert any $\mathbf{d}(\mathsf{tk}, y)$ that is "captured" by QEnc and sk_h will make the scheme forgeable, as a forger can cook up some fake sk_h that might pass the test.)

4.2.1 Target-Restricted Signature Construction

We now present our Target-Restricted Signatures construction. In the next sections we argue its correctness and security based on those of the base PKCom.

Construction 5 (Target-restricted signatures from PKCom) *Suppose we are given a PKCom scheme* $\mathcal{E}^{\mathbf{g},\mathbf{e},\mathbf{d}} := (\mathsf{Key}^{\mathbf{g},\mathbf{e},\mathbf{d}}, \mathsf{Com}^{\mathbf{g},\mathbf{e},\mathbf{d}}, \mathsf{Enc}^{\mathbf{g},\mathbf{e},\mathbf{d}}, \mathsf{Dec}^{\mathbf{g},\mathbf{e},\mathbf{d}})$. *We build an n-target-restricted signature scheme as follows. We assume all the algorithms satisfy the assumption in Note 1.*

- $\mathrm{Gen}^{\mathbf{g},\mathbf{e},\mathbf{d}}(1^\kappa, h)$ *where $h \in [n]$ is the message to be signed. For $i \in [n]$ let* $\mathsf{QGen}_i = \emptyset$.
  1. *For $j \in [n]$, run* $\mathsf{Key}^{\mathbf{g},\mathbf{e},\mathbf{d}}(1^\kappa) \rightarrow (\mathsf{pk}_j, \mathsf{sk}_j)$, *and add all* $\mathbf{g}/\mathbf{e}$ *Q-A pairs to* QGen_j.[11]
  2. *Run* $\mathsf{Com}^{\mathbf{g},\mathbf{e},\mathbf{d}}(\mathsf{pk}_1, \ldots, \mathsf{pk}_n) \rightarrow \mathsf{pp}$ *and add all* $\mathbf{g}/\mathbf{e}$ *Q-A pairs to* QCMP.
  3. *Return* $\mathsf{vrk} = ((\mathsf{pk}_1, \ldots, \mathsf{pk}_n), \cup_{j \neq h} \mathsf{QGen}_j \cup \mathsf{QCMP} \cup \mathsf{QL})$, $\mathsf{sgk} = (\mathsf{sk}_h, \mathsf{QGen}_h)$.
- $\mathrm{Sig}(\mathsf{sgk}, h) \rightarrow \sigma$*: For* sgk *as above, return* $\sigma := (\mathsf{sk}_h, \mathsf{QGen}_h)$.

[11] We do not keep track of $\mathbf{d}$ queries because of Note 1.

– $\mathrm{Ver}^{\mathbf{g},\mathbf{e},\mathbf{d}}(\mathsf{vrk},\sigma,h) = \mathrm{Ver1}(\mathrm{Ver0}^{O}(\mathsf{vrk},h),\sigma)$*: Parse* $\mathsf{vrk} := ((\mathsf{pk}_1,\ldots,\mathsf{pk}_n),\mathsf{S})$ *and* $\sigma := (\mathsf{sk}_h,\mathsf{QGen}_h)$.
  1. $\mathrm{Ver0}^{\mathbf{g},\mathbf{e},\mathbf{d}}(\mathsf{vrk},h) \to \alpha := (\mathsf{vrk},h,m,c,\mathsf{QEnc})$*, where* $(m,c) \leftarrow \mathsf{Enc}^{\mathbf{g},\mathbf{e},\mathbf{d}}(\mathsf{pp},h)$ *and* QEnc *is the set of all Q-A pairs made to* $\mathbf{g}$ *and* $\mathbf{e}$.
  2. $\mathrm{Ver1}(\alpha,\sigma)$*: Retrieve* QEnc *and* S *from* α*. (Recall* $\mathsf{S} = \cup_{j\neq h}\mathsf{QGen}_j \cup \mathsf{QCMP}\cup\mathsf{L}$ *is in* vrk*.) Parse* $\sigma := (\mathsf{sk}_h,\mathsf{QGen}_h)$*. Let* $\mathsf{All} = \mathsf{S}\cup\mathsf{QEnc}\cup\mathsf{QGen}_h$*. Run* $\mathsf{DecSim}(h,\mathsf{sk}_h,\{\mathsf{pk}_i\},c,(\mathsf{All},\mathsf{QEnc},\mathsf{QGen}_h))$*, which simulates the execution of* $\mathsf{Dec}^{O}(h,\mathsf{sk}_h,\{\mathsf{pk}_i\},c)$ *by rendering queries via* $(\mathsf{All},\mathsf{QEnc},\mathsf{QGen}_h)$*, as follows:*
     (a) *For a given* $\mathbf{g}$ *or* $\mathbf{e}$ *query, if the answer is already provided in* All*, reply with that answer; else, with a random string* z *of appropriate length. In case of answering with a random response, add the Q-A pair to* Fake *(initially empty).*[12]
     (b) *For a query* $\mathsf{qu} := ((\mathsf{tk},y) \underset{\mathbf{d}}{\to} ?)$*, if* $(\mathsf{tk} \underset{\mathbf{g}}{\to} \mathsf{ik}) \in \mathsf{QGen}_h \setminus (\mathsf{S}\cup\mathsf{QEnc})$ *and* $((\mathsf{ik},x) \underset{\mathbf{e}}{\to} y) \in (\mathsf{All}\setminus\mathsf{QEnc})\cup\mathsf{Fake}$ *for some* ik *and* x*, respond to* qu *with* x*. Else if* $(\mathsf{tk} \underset{\mathbf{g}}{\to} \mathsf{ik}) \in \mathsf{All}\cup\mathsf{Fake}$ *for some* ik*, and* $((\mathsf{ik},x) \underset{\mathbf{e}}{\to} y) \in \mathsf{All}\cup\mathsf{Fake}$ *for some* x*, respond to* qu *with* x*. Else, respond to* qu *with a random* $r \leftarrow \{0,1\}^{\kappa}$.[13]
     Letting m' *be the output of* DecSim*, output 1 if* $m' = m$ *and 0 otherwise.*

Proof Overview. Our goal is to show that Construction 5 provides both correctness and security. We first discuss correctness. For that we have to argue that if $(\mathsf{sk}_h,\mathsf{QGen}_h)$ are produced honestly, then DecSim run by Ver1 will output m with high probability. For this we have to argue that DecSim respond to all $\mathbf{g}$, $\mathbf{e}$ and $\mathbf{d}$ queries consistently with how they were responded to before (if ever). For example, if a query qu was previously asked during the generation of, say, pk_i, if the same query is asked again by DecSim, it should receive the same response. It is easy to see that this is the case for both $\mathbf{g}$ and $\mathbf{e}$ queries qu. In particular, in Step 2a of Construction 5 we check if qu is in All, which contains all Q-A pairs up to that point. The challenging case is when qu is a $\mathbf{d}$ query: Step 2b Construction 5 responds to $\mathbf{d}$ queries only in some some special cases: in other cases it gives a random response. One scenario in which this happens is when (a) a Q-A pair $((* \underset{\mathbf{g}}{\to} \mathsf{ik})) \in \mathsf{QGen}_h$; and (b) $((\mathsf{ik},*) \underset{\mathbf{e}}{\to} ?)* \in \mathsf{QEnc}$ and (c) $((* \underset{\mathbf{g}}{\to} \mathsf{ik})) \notin \cup_i\mathsf{QGen}_{i\neq h} \cup \mathsf{QCMP} \cup \mathsf{QEnc}$. This means that pp brings some ik information from index h (more specifically, from pk_h). We will prove that the probability that this happens is small; our proof makes use of the fact that $|\mathsf{pp}|$ is compact. In particular, given pp, for at least one index i, pp cannot bring ik information about pk_i that is not present in any other pk_j's. We present and prove the compression statement in Lemma 2. This statement is of independent

[12] Duplicate queries will be replied to with the same random response.
[13] By Note 1, any decryption query is followed by two subsequent $\mathbf{g}$ and $\mathbf{e}$ dummy queries. In the last case where a random response r for (tk,y) is generated, we reply to the subsequent dummy $\mathbf{e}$ query with y.

interest and may find applications in some other impossibility results. Proposition 1 will then make use of this compression theorem to formalize and prove the above statement that pp loses 'ik-information' for some index i. Finally, Lemma 3 uses this proposition to give the correctness proof.

4.2.2 Compression Lemma for TDP Oracles

We present the compression lemma below.

Lemma 2. *Let $\mathcal{A}^{\mathbf{g},\mathbf{e},\mathbf{d}}(1^\kappa) \to z$ be an arbitrary algorithm (not necessarily poly-query) that outputs a string z while calling $\mathbf{O} = (\mathbf{g}, \mathbf{e}, \mathbf{d})$. Let $w := \lceil \frac{2|z|}{3\kappa} + \frac{1}{3} \rceil$. Let $\mathcal{B}^{\mathbf{g},\mathbf{e},\mathbf{d}}(z)$ be an adversary that takes as input z, makes at most $2^\kappa - w$ queries to $\mathbf{g}$ and $\mathbf{d}$ (in total), and an unlimited number of queries to $\mathbf{e}$, and outputs a set $\mathsf{Chal} = \{\mathsf{ik}_1, \ldots, \mathsf{ik}_t\}$, where $w \le t \le 2^{\kappa/3}$. Also, assume B satisfies the assumptions in Note 1. Let Q be the set of all queries/responses made by $\mathcal{B}$. We say Chal is non-trivial if for no $i \in [t]$, $(* \underset{\mathbf{g}}{\to} \mathsf{ik}_i) \in \mathsf{Q}$. We say the event $\mathsf{Success}$ holds if (i) all index keys in Chal are different, (ii) Chal is non-trivial and (iii) for at least w indices $i_1, \ldots, i_w \in [t]$, $\mathsf{ik}_{i_j} \in \mathbf{g}(*)$ for $j \in [w]$.*

We then have $\Pr[\mathsf{Success}] \le 2^{-\kappa/2}$, where the probability is taken over $(\mathbf{g}, \mathbf{e}, \mathbf{d}) \leftarrow \Psi$ and the random coins of $\mathcal{A}$ and $\mathcal{B}$.

Proof. Assume wlog that both $\mathcal{A}$ and $\mathcal{B}$ are deterministic. We will prove the lemma for any fixing of the oracle $\mathbf{e}$ (note that the oracles $\mathbf{g}$ and $\mathbf{e}$ are independent), obtaining a stronger result.

Since both $\mathcal{A}$ and $\mathcal{B}$ are deterministic, for any fixed oracle $\mathbf{g}$ (in addition to $\mathbf{e}$ already fixed) the event Sucess either holds or not; i.e., the probability of $\mathsf{Success}$ is either zero or one with respect to any fixed $\mathbf{g}$. Let $K = 2^\kappa$. We prove that any fixed oracle $\mathbf{g}$ for which $\mathsf{Success}$ holds can be uniquely described with

$$f := \log\left(2^{|z|}\binom{K}{w} w! \binom{t}{w} \frac{(K^3 - w)!}{(K^3 - K)!}\right) \tag{1}$$

bits. This means that there exists at most 2^f different $\mathsf{Successful}$ oracles. Using the inequalities $(a/b)^b \le \binom{a}{b} \le (ae/b)^b$, the fraction of $\mathbf{g}$ oracles for which $\mathsf{Success}$ holds is at most the following.

$$\begin{aligned}
&\frac{2^f}{\text{number of } L \text{ oracles}} \\
&\le \frac{2^{|z|}\binom{K}{w} w! \binom{t}{w} \frac{(K^3-w)!}{(K^3-K)!}}{\frac{K^3!}{(K^3-K)!}} = \frac{2^{|z|}\binom{K}{w}\binom{t}{w}}{\binom{K^3}{w}} \\
&\le \frac{2^{|z|}(\frac{Ke}{w})^w (\frac{te}{w})^w}{(\frac{K^3}{w})^w} = 2^{|z|}(\frac{e^2 t}{K^2 w})^w \le 2^{|z|} w! (\frac{8 \times 2^{\kappa/3}}{2^{2\kappa} w})^w \\
&\le 2^{|z|}(\frac{1}{2^{(3/2)\kappa} w})^w \qquad (\text{because } \frac{8 \times 2^{\kappa/3}}{2^{2\kappa}} \le \frac{1}{2^{(3/2)\kappa}} \text{ for large } \kappa\ ) \\
&\le 2^{|z|}(\frac{1}{2^{(3/2)\kappa}})^w = \frac{1}{2^{(3/2)\kappa w - |z|}} \le \frac{1}{2^{\kappa/2}}.
\end{aligned}$$

The last inequality follows from $\frac{3}{2}kw - |z| \geq k/2$ implied by $w \geq \frac{2|z|}{3\kappa} + \frac{1}{3}$.

We now prove Equation 1. Fix a $\mathsf{Successful}$ oracle $\mathbf{g}$. Let $\mathsf{Chal} = \{\mathsf{ik}_1, \ldots, \mathsf{ik}_t\}$ and wlog assume $\mathsf{ik}_1 <_{\mathsf{lex}} \mathsf{ik}_2 <_{\mathsf{lex}} \cdots <_{\mathsf{lex}} \mathsf{ik}_t$, where $<_{\mathsf{lex}}$ denotes lexicographical ordering. Let $(\mathsf{ik}_{i_1}, \ldots, \mathsf{ik}_{i_w})$ be the w lexicographically smallest elements in Chal that have a pre-image under $\mathbf{g}$, and let $(\mathsf{tk}_{i_1}, \ldots, \mathsf{tk}_{i_w})$ be their pre-images. By Condition (iii) of the lemma such a sequence exists. Let $\mathsf{Chal}_{\mathsf{x}} := (\mathsf{tk}_{i_1}, \ldots, \mathsf{tk}_{i_w})$. Let U be the set of trapdoor keys tk such that $(\mathsf{tk} \underset{\mathbf{g}}{\rightarrow} ?)$ was queried by $\mathcal{B}^{\mathbf{g},\mathbf{e},\mathbf{d}}(z)$. By definition, for any $\mathsf{Successful}$ $\mathbf{g}$, we have $\mathsf{U} \cap \mathsf{Chal}_{\mathsf{x}} = \emptyset$, and hence $\mathsf{U} \subseteq \{0,1\}^\kappa \setminus \mathsf{Chal}_{\mathsf{x}}$.

Given $\mathcal{B}$ we claim that any $\mathsf{Successful}$ oracle $\mathbf{g}$ can be fully described by z, $\mathsf{Chal}_{\mathsf{x}}$, the index set $\{i_1, \ldots, i_w\}$ and the output of $\mathbf{g}$ on all input points in $\{0,1\}^\kappa \setminus \mathsf{Chal}_{\mathsf{x}}$. Indeed, for any $\mathsf{tk} \notin \mathsf{Chal}_{\mathsf{x}}$, the value $\mathbf{g}(\mathsf{tk})$ is already given. We determine the $\mathbf{g}$ outputs on inputs in $\mathsf{Chal}_{\mathsf{x}}$ as follows: Run $\mathcal{B}^{\mathbf{g},\mathbf{e},\mathbf{d}}(z)$ to get Chal. We first explain how to reply to $\mathcal{B}$ queries using the provided information.

1. Answering $\mathbf{g}$ queries of $\mathcal{B}$: Since $\mathsf{U} \subseteq \{0,1\}^\kappa \setminus \mathsf{Chal}_{\mathsf{x}}$ (recall that U contains the set of $\mathcal{B}$'s queries to $\mathbf{g}$) and that $\mathbf{g}$ is fully determined on $\{0,1\}^\kappa \setminus \mathsf{Chal}_{\mathsf{x}}$, we can successfully answer all of $\mathcal{B}$'s $\mathbf{g}$ queries.
2. Answering $\mathbf{e}$ queries of $\mathcal{B}$: the oracle $\mathbf{e}$ is fixed and independent of $\mathbf{g}$.
3. Answering $\mathbf{d}$ queries: for any query $((\mathsf{tk}, y) \underset{\mathbf{d}}{\rightarrow} ?)$, by Note 1, $\mathsf{tk} \in \mathsf{U}$, and hence $\mathsf{tk} \in \{0,1\}^\kappa \setminus \mathsf{Chal}_{\mathsf{x}}$. Thus, the value of $\mathsf{ik} := \mathbf{g}(\mathsf{tk})$ can be determined via the provided information. Once ik is known, since $\mathbf{e}$ is also known, we can compute $\mathbf{d}(\mathsf{tk}, y)$.

Thus, the set Chal can be retrieved. After that, sort its elements lexicographically to get $(\mathsf{ik}_1, \ldots, \mathsf{ik}_t)$, and use the provided indices $(i_1, \ldots, i_w)$ to retrieve $(\mathsf{ik}_{i_1}, \ldots, \mathsf{ik}_{i_w})$. Assuming $\mathsf{Chal}_{\mathsf{x}} = (\mathsf{tk}_1, \ldots, \mathsf{tk}_w)$ we have $\mathbf{g}(\mathsf{tk}_h) = \mathsf{ik}_{i_h}$ for $h \in [w]$. Thus, $\mathbf{g}$ can be reconstructed on inputs in $\mathsf{Chal}_{\mathsf{x}}$, and hence on all inputs.

We now count f the number of bits sufficient to describe $\mathsf{Chal}_{\mathsf{x}}$, the index set $\{i_1, \ldots, i_w\}$ and the output of $\mathbf{g}$ on all of $\{0,1\}^\kappa \setminus \mathsf{Chal}_{\mathsf{x}}$. We can describe the ordered set $\mathsf{Chal}_{\mathsf{x}}$ with $\log(\binom{K}{w} w!)$ bits. For describing the (unordered) index set $\{i_1, \ldots, i_w\}$, note that all the indices are distinct and each is in $[t]$. Thus, we can describe the index set with $\log \binom{t}{w}$ bits. Finally, we can describe the function $\mathbf{g} : \{0,1\}^\kappa \to \{0,1\}^{3\kappa}$ on $\{0,1\}^\kappa \setminus \mathsf{Chal}_{\mathsf{x}}$ with $\log \frac{(K^3-w)!}{(K^3-K)!}$ bits. Equation 1 now follows. □

4.2.3 Proof of Correctness

We now use the compression theorem (Lemma 2) to argue correctness. As explained before (in the proof overview), the goal is to show that pp will lose information, about at least one of pk_i's in the sense below. We first start with some notation.

Definition 5. *We define some notations based on Construction 5.*

1. *For* $\mathsf{vrk} := (\mathsf{pk}_1, \ldots, \mathsf{pk}_n, \ldots)$*, let* Pub_i *for* $i \in [n]$ *be the set of index keys underlying* pk_i*; i.e.,* $\mathsf{ik} \in \mathsf{Pub}_i$ *iff* $(* \underset{\mathbf{g}}{\rightarrow} \mathsf{ik}) \in \mathsf{QGen}_i$
2. *For* $i \in [n]$ *let the random variable* Que_i *be the set of all Q-A pairs during a random execution of* $\mathsf{Enc}^{\mathbf{g},\mathbf{e},\mathbf{d}}(\mathsf{pp}, i)$*. Let* PubC_i *be the set of all* ik *such that (a)* $\mathsf{ik} \in \mathbf{g}(*)$*, (b)* $((\mathsf{ik}, *) \underset{\mathbf{e}}{\rightarrow} *) \in \mathsf{Que}_i$ *and (c)* $(* \underset{\mathbf{g}}{\rightarrow} \mathsf{ik}) \notin \mathsf{Que}_i$*. Let* H_i *be the set of all* ik *such that (a)* $((\mathsf{ik}, *) \underset{\mathbf{e}}{\rightarrow} *) \in \mathsf{Que}_i$ *and (b)* $(* \underset{\mathbf{g}}{\rightarrow} \mathsf{ik}) \notin \mathsf{Que}_i$ *(i.e., same as* PubC_i *except it does not need* ik *to be valid). Note that* $\mathsf{PubC}_i \subseteq \mathsf{H}_i$*.*
3. *Recalling* QEnc *from Construction 5 let* PubC *contain all* ik *such that (a)* $\mathsf{ik} \in \mathbf{g}(*)$*, (b)* $((\mathsf{ik}, *) \underset{\mathbf{e}}{\rightarrow} *) \in \mathsf{QEnc}$*, and (c)* $(* \underset{\mathbf{g}}{\rightarrow} \mathsf{ik}) \notin \mathsf{QEnc}$*. Note that* PubC *and* PubC_h *are identically distributed.*

The following lemma shows that with high probability there exists an index $i \in [n]$ such that during a random execution of $\mathsf{Enc}^{\mathbf{g},\mathbf{e},\mathbf{d}}(\mathsf{pp}, i)$ the set of valid ik's that were not generated in response to $\mathbf{g}$ queries during the same Enc execution and upon which $\mathbf{e}(\mathsf{ik}, *)$ is called, is a subset of $\cup_{j \neq i} \mathsf{Pub}_j$. If this index i happens to be challenge index in Construction 5 (meaning $h = i$), then the simulated decryption DecSim performed by Ver1 will succeed with high probability.

Proposition 1. *Suppose* $\lceil 2\frac{|\mathsf{pp}|}{3\kappa} + \frac{1}{3} \rceil \leq n$*. For random variables as in Definition 5*

$$\Pr[\exists i : (\mathsf{Pub}_i \cap \mathsf{PubC}_i) \subseteq \cup_{j \neq i} \mathsf{Pub}_j] \geq 1 - 2^{-\kappa/2}.$$

Proof. We use notation from Definition 5. Let $t = |\cup_{i=1}^{n} \mathsf{H}_i|$ be the number of distinct index keys in $\cup_{i=1}^{n} \mathsf{H}_i$ and let $w = \lceil 2\frac{|\mathsf{pp}|}{3\kappa} + \frac{1}{3} \rceil$. Let $q = |\cup_{i=1}^{n} \mathsf{PubC}_i|$, and recall all index keys in $\cup_{i=1}^{n} \mathsf{PubC}_i$ are valid. Since for $i \in [n]$, $\mathsf{PubC}_i \subseteq \mathsf{H}_i$, then $\cup_{i=1}^{n} \mathsf{PubC}_i \subseteq \cup_{i=1}^{n} \mathsf{H}_i$. Therefore, based on Lemma 2, $\Pr[q \geq w] \leq 2^{-\kappa/2}$. To see why the former claim holds, view pp in Construction 5 as z in Lemma 2 and view $\cup_{i=1}^{n} \mathsf{H}_i$ as Chal. Let Small be the event that $q < n$. Since, $w = \lceil 2\frac{|\mathsf{pp}|}{3\kappa} + \frac{1}{3} \rceil \leq n$, $\Pr[\mathsf{Small}] \geq 1 - 2^{-\kappa/2}$.

We now show assuming Small holds, there exists an index $i \in [n]$ such that $(\mathsf{Pub}_i \cap \mathsf{PubC}_i) \subseteq \cup_{j \neq i} \mathsf{Pub}_j$. Suppose not. Then, for all $f \in [n]$, there exists a key ik such that $\mathsf{ik}_f \in \mathsf{Pub}_f \cap \mathsf{PubC}_f$ but $\mathsf{ik}_f \notin \cup_{j \neq f} \mathsf{Pub}_j$. We claim that $\mathsf{ik}_1, \mathsf{ik}_2, \ldots, \mathsf{ik}_n$ are distinct values, and since for all j, $\mathsf{ik}_j \in \mathsf{PubC}_j$, the event $\overline{\mathsf{Small}}$ holds, a contradiction. If for two different indices $j, k \in [n]$, $\mathsf{ik}_j = \mathsf{ik}_k$, then $\mathsf{ik}_k \in \mathsf{Pub}_j$ since $\mathsf{ik}_j \in \mathsf{Pub}_j \cap \mathsf{PubC}_j$. This contradicts the assumption that $\mathsf{ik}_k \notin \cup_{j \neq k} \mathsf{Pub}_j$. The bound of the lemma now follows. □

The following proposition shows that if the index h was guessed correctly (i.e., it is the index that pp loses information about the in the sense of the above lemma), then the simulated decryption of DecSim outputs the correct plaintext with high probability.

Proposition 2. *Assuming*

$$(\mathsf{Pub}_h \cap \mathsf{PubC}) \subseteq \cup_{i \neq h} \mathsf{Pub}_i, \tag{2}$$

the probability that the algorithm Ver *(Construction 5) does not output the correct bit is at most* $2^{-2\kappa/3}$*.*

Proof. Let QDec' be the set of all Q-A pairs made to $(\mathbf{g}, \mathbf{e}, \mathbf{d})$ by DecSim inside Ver1. Let

- $\beta_1, \ldots, \beta_l$ be the responses in QDec' sampled uniformly at random to answer encryption queries and let $(\mathsf{ik}_1, x_1), \ldots, (\mathsf{ik}_t, x_l)$ be the corresponding encryption queries.
- Let $\alpha_1, \ldots, \alpha_k$ be the responses in QDec' sampled uniformly at random to answer decryption queries, and let $(\mathsf{tk}_1, y_1), \ldots, (\mathsf{tk}_t, y_t)$ be the corresponding decryption queries.

The output of Ver is $\perp$ only when $\mathsf{T} := \cup_j \mathsf{QGen}_j \cup \mathsf{QCMP} \cup \mathsf{QEnc} \cup \mathsf{QDec}'$ is inconsistent in a TDP sense. Say a point $a \in \{0,1\}^\kappa$ occurs in QEnc if it occurs as the input or output of an $\mathbf{e}$ or $\mathbf{d}$ query: $((*, a) \underset{\mathbf{e}}{\rightarrow} *)$, $((*, *) \underset{\mathbf{e}}{\rightarrow} a)$, $((*, a) \underset{\mathbf{d}}{\rightarrow} *)$ or $((*, *) \underset{\mathbf{d}}{\rightarrow} a)$. First, assume all of α_i and β_j values are distinct. (This happens except with probability $\frac{\mathsf{poly}(\kappa)}{2^\kappa}$.) Assuming this, we might have an inconsistency because the permutation structure is violated, namely when some α_i or β_j occurs in QEnc.[14] The probability of this is also at most $\frac{\mathsf{poly}(\kappa)}{2^\kappa}$. Thus, below assume all of α_i and β_j are distinct, and none of them occur in QEnc.

We might now have an inconsistency because a query response generated by DecSim might conflict with $\mathsf{All} := \cup_j \mathsf{QGen}_j \cup \mathsf{QCMP} \cup \mathsf{QEnc}$. We upperbound the probability of this below.

For a query $\mathsf{qu} := ((\mathsf{tk}, y) \underset{\mathbf{d}}{\rightarrow} ?)$ of DecSim, we might get an inconsistency if

1. for some ik, $(\mathsf{tk} \underset{\mathbf{g}}{\rightarrow} \mathsf{ik}) \in \mathsf{All}$, and
2. $((\mathsf{ik}, x) \underset{\mathbf{e}}{\rightarrow} y) \in \mathsf{All}$, and
3. $x \neq \widetilde{x}$ where $\widetilde{x}$ is the generated response for qu by DecSim.

Call this event Bad. The event Bad indeed causes a conflict because $(\mathsf{tk} \underset{\mathbf{g}}{\rightarrow} \mathsf{ik}) \in \mathsf{All}$ and $((\mathsf{ik}, \widetilde{x}) \underset{\mathbf{e}}{\rightarrow} y) \in \mathsf{All}$ will dictate a response x to a decryption query (tk, y), but a random response $\widetilde{x} \neq x$ was given. Assuming the hypothesis of the lemma (Equation 2) we show whenever Conditions 1 and 2 hold, we will have $(\mathsf{tk} \underset{\mathbf{g}}{\rightarrow} \mathsf{ik}) \in \mathsf{All} \setminus \mathsf{QGen}_h$, and given that $((\mathsf{ik}, x) \underset{\mathbf{e}}{\rightarrow} y) \in \mathsf{All}$, Condition 3 would never hold. (See Step 2b of Ver1 in Construction 5.)

Suppose to the contrary $(\mathsf{tk} \underset{\mathbf{g}}{\rightarrow} \mathsf{ik}) \notin \mathsf{All} \setminus \mathsf{QGen}_h$. Since $(\mathsf{tk} \underset{\mathbf{g}}{\rightarrow} \mathsf{ik}) \in \mathsf{All}$ we conclude $\mathsf{ik} \in \mathsf{PubC}_h$ and $\mathsf{ik} \notin \cup_{i \neq h} \mathsf{PubC}_i$ (Definition 5). Moreover, $((\mathsf{ik}, x) \underset{\mathbf{e}}{\rightarrow} y) \notin \cup_i \mathsf{QGen}_i \cup \mathsf{QCMP}$ because otherwise Condition 3 would never occur. (See Step 2b of Ver1 in Construction 5.) Since $((\mathsf{ik}, x) \underset{\mathbf{e}}{\rightarrow} y) \in \mathsf{All}$ and $\mathsf{All} := \cup_j \mathsf{QGen}_j \cup \mathsf{QCMP} \cup \mathsf{QEnc}$, we get $((\mathsf{ik}, x) \underset{\mathbf{e}}{\rightarrow} y) \in \mathsf{QEnc}$. Since we assumed $(\mathsf{tk} \underset{\mathbf{g}}{\rightarrow} \mathsf{ik}) \notin \mathsf{All} \setminus \mathsf{QGen}_h$, we get $(\mathsf{tk} \underset{\mathbf{g}}{\rightarrow} \mathsf{ik}) \notin \mathsf{QEnc}$. Thus, $\mathsf{ik} \in \mathsf{PubC}$. We have established $\mathsf{ik} \in \mathsf{PubC}_h \cap \mathsf{PubC}$, and $\mathsf{ik} \notin \cup_{i \neq h} \mathsf{PubC}_i$, contradicting Equation 2.

□

[14] This case does not necessarily lead to an inconsistency, but we show it will, nonetheless, occur with small probability.

Lemma 3 (Correctness). *Suppose Π is the signature scheme defined in Construction 5 with oracle access of the form $(\mathrm{Gen}^{\mathbf{g},\mathbf{e},\mathbf{d}}, \mathrm{Sig}, \mathrm{Ver}^{\mathbf{g},\mathbf{e},\mathbf{d}})$ and the PKCom scheme underlying Π is $(1-\epsilon)$-correct. Then, Π is $(1-\epsilon)\frac{(1-2^{-\kappa/3})}{n}$-correct.*

Proof. Let $\mathsf{Success}$ be the event that the verification algorithm outputs the correct bit. In other words, if $\mathsf{Success}_i$ holds, we have: $\mathrm{Ver}^{\mathbf{g},\mathbf{e},\mathbf{d}}(\mathsf{vrk}, i, \mathrm{Sig}(\mathsf{sgk}, i)) = 1$ where $(\mathsf{sgk}, \mathsf{vrk}) \leftarrow \mathrm{Gen}^{\mathbf{g},\mathbf{e},\mathbf{d}}(1^\kappa, i)$. Finally, let X be the random variable denoting the message chosen to be signed. Also, let Good be the event that there exists $h \in [n]$ for which Proposition 1 holds. In other words, if Good happens, we have:

$$(\mathsf{Pub}_h \cap \mathsf{PubC}) \subseteq \cup_{j \neq h} \mathsf{Pub}_j$$

Based on Proposition 1, $\Pr[\mathsf{Good}] \geq 1 - 2^{-\kappa/2}$. Therefore:

$$\Pr[\Pi \text{ is correct}] = \sum_{i=1}^{n} \Pr[\mathsf{Success}|\mathsf{X} = i] \Pr[\mathsf{X} = i] = \frac{1}{n} \sum_{i=1}^{n} \Pr[\mathsf{Success}_i]$$

$$\Pr[\mathsf{Success}_i] = \Pr[\mathsf{Success}_i|\mathsf{Good}] \Pr[\mathsf{Good}] + \Pr[\mathsf{Success}_i|\overline{\mathsf{Good}}] \Pr[\overline{\mathsf{Good}}] \geq$$

$$\Pr[\mathsf{Success}_i|\mathsf{Good}] \Pr[\mathsf{Good}] \geq (1 - 2^{-\kappa/2}) \Pr[\mathsf{Success}_i|\mathsf{Good}]$$

$$\Rightarrow Pr[\Pi \text{ is correct}] = \frac{1}{n} \sum_{i=1}^{n} \Pr[\mathsf{Success}_i] \geq \frac{1}{n}(1-2^{-\kappa/2}) \sum_{i=1}^{n} \Pr[\mathsf{Success}_i|\mathsf{Good}] \geq$$

$$\frac{1}{n}(1 - 2^{-\kappa/2}) \Pr[\mathsf{Success}_h|\mathsf{Good}] \geq \frac{1}{n}(1 - 2^{-\kappa/2})(1 - 2^{-2\kappa/3}) \geq \frac{(1 - 2^{-\kappa/3})}{n}$$

□

4.2.4 Proof of Security

Now, we prove one-time unforgeability of Construction 5. We first present the following standard bound.

Lemma 4. *Let $X_1, \ldots, X_{t+1}$ be independent, Bernoulli random variables, where $\Pr[X_i = 1] = p$, for all $i \leq t+1$. Then*

$$\Pr[X_1 = 0 \wedge \cdots \wedge X_t = 0 \wedge X_{t+1} = 1] \leq \frac{1}{t}.$$

Lemma 5 (Security of Construction 5). *Construction 5 is one-time unforgeable if the PKCom scheme is secure.*

Before delving in the proof, let us sketch the main challenges in the proof and how we overcome them.

Proof sketch of Lemma 5 Suppose $\mathcal{B}$ is successful in forging a signature in $\mathcal{E}^{\mathbf{g},\mathbf{e},\mathbf{d}}$. We need to argue how to build an adversary $\mathcal{A}^{\mathcal{B},\mathbf{g},\mathbf{e},\mathbf{d}}$ against PKCom. The adversary $\mathcal{B}$ forges a signature by presenting $\sigma := (\mathsf{sk}_h, \mathsf{QGen}_h)$. Since $\mathcal{B}$ forges a signature, we know that $\mathsf{DecSim}(h, \mathsf{sk}_h, \{\mathsf{pk}_i\}, c, (\mathsf{All}, \mathsf{QEnc}, \mathsf{QGen}_h))$, outputs the correct plaintext bit for c. Thinking of c as a challenge ciphertext for $\mathcal{A}$, the adversary $\mathcal{A}$ does not have QEnc so to be able to run

$$\mathsf{DecSim}(h, \mathsf{sk}_h, \{\mathsf{pk}_i\}, c, (\mathsf{All}, \mathsf{QEnc}, \mathsf{QGen}_h)).$$

Instead, $\mathcal{A}$ has full access to all the oracles $\mathbf{g}, \mathbf{e}, \mathbf{d}$. The reduction works by letting $\mathcal{A}$ run $\mathsf{DecSim}(h, \mathsf{sk}_h, \{\mathsf{pk}_i\}, c, (\mathsf{All}, \mathsf{QEnc}, \mathsf{QGen}_h))$, but those queries which require knowledge of QEnc to be responded to, will be answered by consulting the real oracles $(\mathbf{g}, \mathbf{e}, \mathbf{d})$. (Recall that DecSim does not have oracle access to $(\mathbf{g}, \mathbf{e}, \mathbf{d})$.) Our analysis will show that the entire distribution of all Q-A pairs underlying all public keys, ciphertexts and those that appear during decryption is indistinguishable in the two worlds: the first world is one where things are decrypted as per DecSim by using QEnc, and the second world is one where the real oracles $(\mathbf{g}, \mathbf{e}, \mathbf{d})$ are used for decryption, as explained above. Let us highlight some of the challenges and how we overcome them.

Suppose that $\mathcal{B}$ makes a dummy query $(\mathsf{tk} \underset{\mathbf{g}}{\rightarrow} ?)$ for a random tk and puts (x, tk) in its forged secret key sk_h, where x is the first half of ik. And suppose the decryption algorithm on sk_h always call the query $\mathsf{qu} := (\mathsf{tk} \underset{\mathbf{g}}{\rightarrow} ?)$. Now assuming that $(\mathsf{tk} \underset{\mathbf{g}}{\rightarrow} ?)$ is not a query in QEnc (which is likely as qu is chosen randomly), under DecSim this query is responded to with random (which is independent of x), while under the above strategy, of consulting the real oracle $\mathbf{g}$, we respond to with ik itself (whose first half equals x). Thus, the two distributions become distinguishable. To get around this, we use the oracles $(\mathbf{g}, \mathbf{e}, \mathbf{d})$ in a controlled way. Namely, we maintain a set of Q-A pairs $\mathsf{collect}$, and if a given query is not answered in $\mathsf{collect}$ we then consult the real oracles $(\mathbf{g}, \mathbf{e}, \mathbf{d})$. To sample $\mathsf{collect}$, we run $(c', *) \leftarrow \mathsf{Enc}^{\mathbf{g},\mathbf{e},\mathbf{d}}(\mathsf{pp}, h)$ many times, each time recording the queries in a fresh local set QEnc', and then run DecSim on sk_h and c' while using QEnc' for decryption. We collect all the Q-A pairs that occur during the executions of DecSim in the set $\mathsf{collect}$. It can now be seen with this enhancement the above issue goes way: both worlds will respond to qu randomly.

Proof. We show how to transform an adversary $\mathcal{B}$ against the signatures into an adversary $\mathcal{A}^{\mathcal{B},\mathbf{g},\mathbf{e},\mathbf{d}}$ against PKCom.

1. $\mathcal{A}$ samples h, and for $i \in [n] \setminus \{h\}$, it samples $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{Key}(1^\kappa)$ and collects the Q-A pairs in QGen_i. It then gives $(\mathsf{pk}_i, \mathsf{sk}_i)_{i \neq h}$ to the challenger to receive $(\mathsf{pk}, \mathsf{ct})$.
2. $\mathcal{A}$ forms $\mathsf{vrk} := (\{\mathsf{pk}_i\}, \cup_{j \neq h} \mathsf{QGen}_j \cup \mathsf{QCMP})$, where QCMP is the set of Q-A pairs made during $\mathsf{Com}(\mathsf{pk}_1, \ldots, \mathsf{pk}_n)$.
3. $\mathcal{A}$ runs $(\mathsf{sk}'_h, \mathsf{QGen}'_h) \leftarrow \mathcal{B}(\mathsf{vrk}, h)$.
4. $\mathcal{A}$ runs $(\mathsf{ct}', *) \leftarrow \mathsf{Enc}^{\mathbf{g},\mathbf{e},\mathbf{d}}(\mathsf{pp}, h)$ η times, each time recording the queries in a set QEnc', and then runs DecSim while using QEnc', and records all queries in $\mathsf{collect}$.

5. $\mathcal{A}$ runs $\mathsf{DecSim}_0(\mathsf{sk}'_h, h, \mathsf{ct})$: Let $\mathsf{All}' = \cup_{j\neq h}\mathsf{QGen}_j \cup \mathsf{QCMP} \cup \mathsf{collect}$. For any $\mathbf{g}/\mathbf{e}$ query, if already answered in $\mathsf{QGen}'_h \cup \mathsf{All}'$, use that answer; else, reply with calling the $\mathbf{g}/\mathbf{e}$ oracle on the same query. For $\mathsf{qu} := ((\mathsf{tk}, y) \underset{\mathbf{d}}{\rightarrow} ?)$:
   (a) if for some ik and x
      i. $(\mathsf{tk} \underset{\mathbf{g}}{\rightarrow} \mathsf{ik}) \in \mathsf{All}'$ and $((\mathsf{ik}, x) \underset{\mathbf{e}}{\rightarrow} y) \in \mathsf{All}' \cup \mathsf{QGen}'_h \backslash \mathsf{collect}$, respond to qu with x.
      ii. if $(\mathsf{tk} \underset{\mathbf{g}}{\rightarrow} \mathsf{ik}) \in \mathsf{QGen}'_h \setminus \mathsf{All}'$ and $((\mathsf{ik}, x) \underset{\mathbf{e}}{\rightarrow} y) \in \mathsf{All}' \cup \mathsf{QGen}'_h \backslash \mathsf{collect}$, respond to qu with x.
      iii. if $(\mathsf{tk} \underset{\mathbf{g}}{\rightarrow} \mathsf{ik}) \in \mathsf{QGen}'_h \setminus \mathsf{All}'$ and $((\mathsf{ik}, x) \underset{\mathbf{e}}{\rightarrow} y) \notin \mathsf{All}' \cup \mathsf{QGen}'_h \backslash \mathsf{collect}$, respond to qu with random.
   (b) Else, respond to qu with $\mathbf{d}(\mathsf{tk}, y)$.

Let $\mathsf{All} := \cup_{j\neq h}\mathsf{QGen}_j \cup \mathsf{QCMP} \cup \mathsf{QEnc}$ where QEnc is the set of all Q-A pairs made during $(\mathsf{ct}, *) \leftarrow \mathsf{Enc}(\mathsf{pp}, h; R)$. Also let QDec and QDec' be the set of Q-A pairs made by DecSim and DecSim_0, respectively and let T be the set of all Q-A pairs made by $\mathcal{B}$ while forging $(\mathsf{sk}'_h, \mathsf{QGen}'_h)$. Consider the following two distributions:

$$\mathsf{D} : (\mathsf{QGen}_{j\in[n]} \cup \mathsf{QCMP}, \{\mathsf{pk}_j\}_{j\in[n]}, \mathsf{QEnc}, \mathsf{ct}, \sigma, \mathsf{QDec}, \mathsf{T}, R, \mathsf{sk}'_h, \mathsf{QGen}'_h)$$

and

$$\mathsf{D}' : (\mathsf{QGen}_{j\in[n]} \cup \mathsf{QCMP}, \{\mathsf{pk}_j\}_{j\in[n]}, \mathsf{QEnc}, \mathsf{ct}, \sigma, \mathsf{QDec}', \mathsf{T}, R, \mathsf{sk}'_h, \mathsf{QGen}'_h).$$

For each key pair $(\mathsf{tk} \rightarrow \mathsf{ik}) \in \mathsf{T} \cup \mathsf{QGen}_h$, we define $\eta + 1$ variables as below: For $1 \leq i \leq \eta$, let X_i be a Bernoulli variable that equals 1 when $(\mathsf{tk} \underset{\mathbf{g}}{\rightarrow} *)$ appears in ith iteration of collecting queries in $\mathsf{collect}$ at step 4 above and 0 otherwise. Also, let $X_{\eta+1} = 1$ if $(\mathsf{tk} \underset{\mathbf{g}}{\rightarrow} *)$ appears in execution of $\mathsf{DecSim}(\mathsf{sk}'_h, h, \mathsf{ct})$ while running Ver_1 and 0 otherwise. $X_1, \ldots, X_{\eta+1}$ are independent and have the same probability of being 1. This is because all X_i, for $1 \leq i \leq \eta$, are output of the same randomized experiment and $X_{\eta+1}$ can also be seen as a result of the same experiment. Let QDec_{key} be the set of all tk such that $(\mathsf{tk} \underset{\mathbf{g}}{\rightarrow} *)$ is queried in execution of $\mathsf{DecSim}(\mathsf{crs}, h, \sigma, \mathsf{vrk}, c)$ while running Ver_1. Also, let $\mathsf{collect}_{key}$ be the set of all tk such that $(\mathsf{tk} \underset{\mathbf{g}}{\rightarrow} *) \in \mathsf{collect}$.

Therefore, based on Lemma 4, for any key pair $(\mathsf{tk} \rightarrow \mathsf{ik}) \in \mathsf{T} \cup \mathsf{QGen}_h$,

$$\Pr[\mathsf{tk} \in \mathsf{QDec}_{key}/\mathsf{collect}_{key}] = \Pr[X_1 = 0 \wedge \cdots \wedge X_\eta = 0 \wedge X_{\eta+1} = 1] \leq \frac{1}{\eta}. \quad (3)$$

Therefore, the probability that for some $(\mathsf{tk} \underset{\mathbf{g}}{\rightarrow} \mathsf{ik}) \in \mathsf{QDec}$, $(\mathsf{tk} \underset{\mathbf{g}}{\rightarrow} \mathsf{ik}) \notin \mathsf{collect}$ but $\mathbf{g}(\mathsf{tk})$ is queried while running Ver_1 is at most $\frac{\gamma}{\eta}$ where $\gamma = \mathsf{poly}(\kappa)$ is the number of queries in $\mathsf{T} \cup \mathsf{QGen}_h$. Let $\eta = \gamma^2$; thus, with probability at least $1 - \frac{1}{\gamma}$,

for any key pair $(\mathsf{tk} \underset{\mathbf{g}}{\rightarrow} \mathsf{ik}) \in (\mathsf{T} \cup \mathsf{QGen}_h) \cap \mathsf{QDec}$, we also have $(\mathsf{tk} \underset{\mathbf{g}}{\rightarrow} \mathsf{ik}) \in \mathsf{collect}$.

Suppose the decryption algorithm makes no duplicate queries. We first prove that with high probability, for all $\mathbf{g}$ type queries, both distributions are the same. Suppose not and let $(\mathsf{tk}^* \underset{\mathbf{g}}{\rightarrow} ?)$ be the first query for which DecSim and DecSim_0 will answer with differently and make the distributions different. $(\mathsf{tk}^* \underset{\mathbf{g}}{\rightarrow} *)$ cannot be present in $\mathsf{QGen}'_h \cup \mathsf{All}'$ because otherwise both DecSim and DecSim_0 would answer with the same response. Also, $(\mathsf{tk}^* \underset{\mathbf{g}}{\rightarrow} *)$ cannot appear in QEnc because otherwise DecSim will answer according to QEnc and DecSim_0 will query the oracle which results in DecSim and DecSim_0 answering with the same response as the oracle is correct. That means that $(\mathsf{tk}^* \underset{\mathbf{g}}{\rightarrow} *) \in \mathsf{T} \cup \mathsf{QGen}_h$ but based on the above conclusion, since $\mathsf{tk} \in \mathsf{QDec}_{key}$ with probability at least $1 - \frac{1}{\gamma}$, $(\mathsf{tk}^* \underset{\mathbf{g}}{\rightarrow} *)$ has been collected in $\mathsf{collect}$, therefore $(\mathsf{tk}^* \underset{\mathbf{g}}{\rightarrow} *) \in \mathsf{All}'$ which is a contradiction.

Now, we can similarly prove that with high probability, for all $\mathbf{e}$ type queries, both distributions are the same. Suppose not and let $((\mathsf{ik}^*, x^*) \underset{\mathbf{e}}{\rightarrow} ?)$ be the first query for which DecSim and DecSim_0 will answer with differently and make the distributions different. $((\mathsf{ik}^*, x^*) \underset{\mathbf{e}}{\rightarrow} *)$ cannot be in $\mathsf{QGen}'_h \cup \mathsf{All}' \cup \mathsf{QEnc}$ because otherwise both DecSim and DecSim_0 will answer with the same response. That means that $((\mathsf{ik}^*, x^*) \underset{\mathbf{e}}{\rightarrow} *) \in \mathsf{T} \cup \mathsf{QGen}_h$. Similar to 3, we can prove that with probability at least $1 - \frac{1}{\gamma}$, for any pair (ik, x) where $((\mathsf{ik}, x) \underset{\mathbf{e}}{\rightarrow} *) \in (\mathsf{T} \cup \mathsf{QGen}_h) \cap \mathsf{QDec}$, we also have $((\mathsf{ik}, x) \underset{\mathbf{e}}{\rightarrow} *) \in \mathsf{collect}$.

Therefore, since $((\mathsf{ik}^*, x^*) \underset{\mathbf{e}}{\rightarrow} *) \in \mathsf{QDec}$ with probability at least $1 - \frac{1}{\gamma}$, $((\mathsf{ik}^*, x^*) \underset{\mathbf{e}}{\rightarrow} *)$ has been collected in $\mathsf{collect}$. Thus, $((\mathsf{ik}^*, x^*) \underset{\mathbf{e}}{\rightarrow} *) \in \mathsf{All}'$ which is a contradiction.

We finally prove that with high probability, for all $\mathbf{d}$ type queries, both distributions are the same. Let $\mathsf{qu} := ((\mathsf{tk}, y) \underset{\mathbf{d}}{\rightarrow} ?)$ be the first decryption query that makes the distributions different. There are five cases:

1. There exists a ik such that $(\mathsf{tk} \underset{\mathbf{g}}{\rightarrow} \mathsf{ik}) \in \mathsf{All} \cup \mathsf{QGen}'_h \backslash \mathsf{QEnc}$ and there exists x such that $((\mathsf{ik}, x) \underset{\mathbf{e}}{\rightarrow} y) \in \mathsf{QGen}'_h \cup \mathsf{All} \backslash \mathsf{QEnc}$: In this case, both distributions will answer with x.
2. There exists a ik such that $(\mathsf{tk} \underset{\mathbf{g}}{\rightarrow} \mathsf{ik}) \in \mathsf{All}$ and there exists x such that $((\mathsf{ik}, x) \underset{\mathbf{e}}{\rightarrow} y) \in \mathsf{QEnc}$: In this case, DecSim will answer with x while DecSim_0 will query the oracle and answer accordingly. However, since QEnc is produced by the same oracle and the oracle is correct, the the oracle's answer will be x and both distribution will answer the same.
3. There exists a ik such that $(\mathsf{tk} \underset{\mathbf{g}}{\rightarrow} \mathsf{ik}) \in \mathsf{QEnc}$ and there exists x such that $((\mathsf{ik}, x) \underset{\mathbf{e}}{\rightarrow} y) \in \mathsf{All}$: In this case, DecSim will answer with x while DecSim_0

will query the oracle and answer accordingly. Therefore, similar to Item 2 both distribution will answer the same.

4. There exists a ik such that $(\mathsf{tk} \underset{\mathbf{g}}{\rightarrow} \mathsf{ik}) \in \mathsf{QGen}'_h \backslash \mathsf{All}$ and there exists no x such that $((\mathsf{ik}, x) \underset{\mathbf{e}}{\rightarrow} y) \in \mathsf{QGen}'_h \cup \mathsf{All} \backslash \mathsf{QEnc}$: In this case, both distributions will answer with a random response.
5. There exists a ik such that $(\mathsf{tk} \underset{\mathbf{g}}{\rightarrow} \mathsf{ik}) \in \mathsf{All}$ but for no x, $((\mathsf{ik}, x) \underset{\mathbf{e}}{\rightarrow} y) \in \mathsf{QGen}'_h \cup \mathsf{All}$: In this case, if $((\mathsf{ik}, x) \underset{\mathbf{e}}{\rightarrow} y) \notin \mathsf{collect}$, DecSim will answer with a random response while DecSim_0 will query oracle and answer accordingly. However, note that for some y, $((\mathsf{ik}, x) \underset{\mathbf{e}}{\rightarrow} y)$ must appear in $\mathsf{QGen}_h \cup \mathsf{T}$ because otherwise there is no way for a distinguisher who only has access to the two distributions to differentiate between the random and true response. Similar to 3, we can prove that with probability at least $1 - \frac{1}{\gamma}$, for any pair (ik, x) where $((\mathsf{ik}, x) \underset{\mathbf{e}}{\rightarrow} *) \in (\mathsf{T} \cup \mathsf{QGen}_h) \cap \mathsf{QDec}$, we also have $((\mathsf{ik}, x) \underset{\mathbf{e}}{\rightarrow} *) \in \mathsf{collect}$.

   Therefore, since $((\mathsf{ik}, x) \underset{\mathbf{e}}{\rightarrow} y) \in \mathsf{QDec}$ with probability at least $1 - \frac{1}{\gamma}$, $((\mathsf{ik}, x) \underset{\mathbf{e}}{\rightarrow} y)$ has been collected in $\mathsf{collect}$. Thus, with probability at least $1 - \frac{1}{\gamma}$, DecSim_0 will answer with a random response and both distribution will be the same.

□

5 Impossibility in Shoup's Generic Group Model

In this section, we show that there exists a Shoup's GGM oracle relative to which PKCom does not exist. First, we recall Shoup's generic group model [Sho97].

Definition 6. *Let $p \in \mathbb{N}$ be a positive integer and let $S = \{0,1\}^w$ be a set of strings where $w \geq \log p + \kappa$. A random injection $\mathbf{label} : \mathbb{Z}_p \to S$ is chosen, which we will call the labeling function. All parties have access to the oracle $\mathbb{G}_{RR} = (\mathbf{label}, \mathbf{add})$, defined in the following way.*

- **label***: The party submits $x \in \mathbb{Z}_p$, and receives the label of x.*
- **add***: The party submits $(\ell_1, \ell_2) \in S^2$. If there exists $x_1, x_2 \in \mathbb{Z}_p$ such that $\mathbf{label}(x_1) = \ell_1$ and $\mathbf{label}(x_2) = \ell_2$, then the party receives $\mathbf{label}(x_1 + x_2)$. Otherwise, the party receives $\perp$. Note that* **label** *completely determines* **add** *and thus also determines the whole oracle.*

In this section, we will assume that $p \in [2^\kappa, 2^{\kappa+1}]$ and $S = \{0,1\}^{3\kappa}$.

Lemma 6. *Let $\mathcal{A}^{\mathbf{label},\mathbf{add}}(1^\kappa) \to z$ be an arbitrary algorithm (not necessarily poly-query) that outputs a string z while calling $\mathbb{G}_{RR} = (\mathbf{label}, \mathbf{add})$. Let*

$\mathcal{B}^{\mathbf{label},\mathbf{add}}(z)$ *be an adversary that takes as input the advice* z*, makes at most* u *queries to* **label** *and* **add** *in total, and outputs a set of labels* $\mathsf{Chal} = \{\ell_1, \ldots, \ell_t\}$ *where* $t = 2^{\kappa/3}$*. Let* $w := \lceil \frac{2(|z|+u)}{3\kappa} + \frac{1}{3} \rceil$*. Let* Q *be the set of all labels that appear in the responses to the queries made by* $\mathcal{B}$*. We say the event* $\mathsf{Success}$ *holds if (i) all* ℓ_i*'s are different, (ii) for no* $i \in [t]$*,* $\ell_i \in Q$*, and (iii) for at least* w *indices* $i_1, \ldots, i_w \in [t]$*,* $\ell_{i_j} \in \mathbf{label}(*)$ *for* $j \in [w]$*. We then have* $\Pr[\mathsf{Success}] \leq 2^{-\kappa/2} = \mathrm{negl}(\kappa)$*, where the probability is taken over* $L \leftarrow \Psi$ *and the random coins of* $\mathcal{A}$ *and* $\mathcal{B}$*.*

The proof of Lemma 6 is very similar to the proof of Lemma 2 and thus it is moved to the appendix.

5.1 Impossibility of CRS-free PKCom in Shoup's GGM

Similar to Section 4.2, we first present the transformation to target-restricted signatures for the case in which the PKCom does not have a CRS.

Definition 7. *Let* x_ℓ *be the variable that is either* $\perp$ *or* x *where* x *is the element in* $\mathbb{Z}_p$ *whose label is* ℓ*. If* ℓ *is invalid label, then* x_ℓ *is* $\perp$*.*

Definition 8. *Suppose* Q *is a set of group operation Q-A pairs. We define* $\mathsf{Eq}(\mathsf{Q})$ *to be the set of homogeneous linear equations that are directly implied by* Q*. In other words, for a query* $((\ell_1, \ell_2) \xrightarrow[\mathbf{add}]{} \ell_3) \in \mathsf{Q}$*, if* $\ell_3 \neq \perp$*, we add to* $\mathsf{Eq}(\mathsf{Q})$ *the equation* $x_{\ell_1} + x_{\ell_2} - x_{\ell_3} = 0$*.*

Definition 9. *For a set of Q-A pairs* Q*, define* $\mathsf{Var}(\mathsf{Q})$ *to be the set of all labels* $\ell \neq \perp$ *such that* $((*, *) \xrightarrow[\mathbf{add}]{} \ell) \in \mathsf{Q}$*.*

Definition 10. *Let* LS *be the set of all possible labels* $\ell \in S = \{0,1\}^*$ *such that* $\ell = \mathbf{label}(x)$ *for some* $x \in \mathbb{Z}_p$*. Also, let* $\upsilon = \mathbf{label}(1)$*.*

Definition 11 (Updating the Known function). *Given a list* L *of* **add** *Q-A pairs, update* $\mathsf{Known} \leftarrow \mathsf{Upd}(L)$ *as follows. Do the following until no further updates are possible: if there exists* $((\ell_1, \ell_2) \xrightarrow[\mathbf{add}]{} \ell_3) \in L$ *such that* $\mathsf{Known}(\ell_1) = \top$ *or* $\mathsf{Known}(\ell_2) = \top$*, update* $\mathsf{Known}(\ell_3) = \top$*.*

Theorem 6. *If there exists a* $(1-\epsilon)$*-correct PKCom scheme* $(\mathsf{Key}^{\mathbb{G}_{RR}}, \mathsf{Com}^{\mathbb{G}_{RR}}, \mathsf{Enc}^{\mathbb{G}_{RR}}, \mathsf{Dec}^{\mathbb{G}_{RR}})$ *in the RR generic group model, then there exists a* δ*-correct target-restricted signature scheme in the same model where* $\delta = (1-\epsilon)\frac{(1-2^{-\kappa/3})}{n}$*.*

Construction 7 *Let* $(\mathsf{Key}^{\mathbb{G}_{RR}}, \mathsf{Com}^{\mathbb{G}_{RR}}, \mathsf{Enc}^{\mathbb{G}_{RR}}, \mathsf{Dec}^{\mathbb{G}_{RR}})$ *be a PKCom scheme. We will assume all queries made to* $\mathbb{G}_{RR}$ *are* **add** *queries since* **label** *queries can be answered using* **add** *queries given* $\upsilon = \mathbf{label}(1)$*. We construct a target-restricted signature scheme defined over messages in* $[n]$ *from the PKCom scheme. We let* $\mathsf{Known}\colon \mathsf{LS} \to \{\perp, \top\}$*, initially set to* $\mathsf{Known}(\upsilon) = \top$*, and* $\perp$ *for all other labels.*

– $\mathrm{Gen}^{\mathbb{G}_{RR}}(1^\kappa, h) \to (\mathsf{sgk}, \mathsf{vrk})$ *where* $h \in [n]$ *is the message to be signed. For* $i \in [n]$ *let* $\mathsf{QGen}_i = \emptyset$.
  1. *For* $1 \le j \le n$, *run* $\mathsf{Key}^{\mathbb{G}_{RR}}(1^\kappa) \to (\mathsf{pk}_j, \mathsf{sk}_j)$ *and add all Q-A pairs made to* $\mathbb{G}_{RR}$ *to* QGen_j.
  2. *Run* $\mathsf{Com}^{\mathbb{G}_{RR}}(\mathsf{pk}_1, \ldots, \mathsf{pk}_n) \to \mathsf{pp}$ *and let* QCMP *be the set of all Q-A pairs made to* $\mathbb{G}_{RR}$.
  3. *Update* $\mathsf{Known} \leftarrow \mathsf{Upd}(\cup_{i \neq h} \mathsf{QGen}_i \cup \mathsf{QCMP})$ *(Definition 11).*
  4. *Return* $\mathsf{vrk} = ((\mathsf{pk}_1, \ldots, \mathsf{pk}_n), \cup_{j \neq h} \mathsf{QGen}_j \cup \mathsf{QCMP}, \mathsf{Known}, \upsilon)$, $\mathsf{sgk} = (\mathsf{sk}_h, \mathsf{QGen}_h)$.

– $\mathrm{Sig}(\mathsf{sgk}, h) \to \sigma$*: For* sgk *as above, return* $\sigma := (\mathsf{sk}_h, \mathsf{QGen}_h)$.

– $\mathrm{Ver}^{\mathbb{G}_{RR}}(\mathsf{vrk}, \sigma, h) = \mathrm{Ver1}(\mathrm{Ver0}^{\mathbb{G}_{RR}}(\mathsf{vrk}, h), \sigma)$: *Parse* $\mathsf{vrk} := ((\mathsf{pk}_1, \ldots, \mathsf{pk}_n), \mathsf{A}, \mathsf{Known}, \upsilon)$ *and* $\sigma := (\mathsf{sk}_h, \mathsf{QGen}_h)$.
  1. $\mathrm{Ver0}^{\mathbb{G}_{RR}}(\mathsf{vrk}, h) \to \alpha := (\mathsf{vrk}, h, m, c, \mathsf{QEnc})$, *where* $(m, c) \leftarrow \mathsf{Enc}^{\mathbb{G}_{RR}}(\mathsf{pp}, h)$ *and* QEnc *is the set of all Q-A pairs made to* $\mathbb{G}_{RR}$.
  2. $\mathrm{Ver1}(\alpha, \sigma)$: *Retrieve* QEnc, A *and* Known *from* α. *Recall* $\mathsf{A} = \cup_{j \neq h} \mathsf{QGen}_j \cup \mathsf{QCMP}$. *Update* $\mathsf{Known} \leftarrow \mathsf{Upd}(\mathsf{QEnc})$. *Let* $\mathsf{All} = \cup_{j \neq h} \mathsf{QGen}_j \cup \mathsf{QCMP} \cup \mathsf{QEnc}$. *Run* DecSim *which simulates the execution of* $\mathsf{Dec}^{\mathbb{G}_{RR}}(h, \mathsf{sk}_h, \{\mathsf{pk}_i\}, c)$ *by rendering queries via* $(\mathsf{All}, \mathsf{QGen}_h)$, *as follows: Initialize two sets* $\mathsf{E} = \mathsf{Eq}(\mathsf{All})$ *and* $\mathsf{V} = \mathsf{Var}(\mathsf{All})$. *For a given query* $\mathbf{add}(\ell_1, \ell_2)$ *do the following:*
     (a) *If* $\ell_1 \notin \mathsf{V} \cup \mathsf{Var}(\mathsf{QGen}_h)$ *or* $\ell_2 \notin \mathsf{V} \cup \mathsf{Var}(\mathsf{QGen}_h)$, *respond to the query with* $\perp$.
     (b) *Else if both* $\ell_1, \ell_2 \in \mathsf{V}$, *if there exists* $\ell \in \mathsf{V} \cup \mathsf{Var}(\mathsf{QGen}_h)$ *such that* $x_{\ell_1} + x_{\ell_2} - x_\ell \in \mathsf{Span}(\mathsf{E} \cup \mathsf{Eq}(\mathsf{QGen}_h))$, *return* ℓ. *If no such an* ℓ *is found, respond with a random label* ℓ', *add* $x_{\ell_1} + x_{\ell_2} - x_{\ell'}$ *to* E *and add* ℓ' *to* V. *Also, set* $\mathsf{Known}(\ell') = \top$.
     (c) *Else if there exists a label* ℓ *such that* $x_{\ell_1} + x_{\ell_2} - x_{\ell'} \in \mathsf{Span}(\mathsf{Eq}(\mathsf{QCMP} \cup_i \mathsf{QGen}_i))$, *return* ℓ;
     (d) *Else, if there exists a label* ℓ *such that* $\mathsf{Known}(\ell) = \top$ *and* $x_{\ell_1} + x_{\ell_2} - x_\ell \in \mathsf{Span}(\mathsf{E} \cup \mathsf{Eq}(\mathsf{QGen}_h))$, *return* ℓ. *Else, respond with a random label* ℓ', *add* $x_{\ell_1} + x_{\ell_2} - x_{\ell'}$ *to* E, *and add* ℓ' *to* V. *Also, set* $\mathsf{Known}(\ell') = \top$.

  Let m' *be the output of* DecSim, *output 1 if* $m = m'$ *and 0 otherwise.*

Definition 12. *We define some notations based on Construction 7.*

1. *For* $i \in [n]$ *let the random variable* Qu_i *be the set of all Q-A pairs during a random execution of* $\mathsf{Enc}^{\mathbb{G}_{RR}}(\mathsf{pp}, i)$. *Let* Q_i *be the set of all labels* ℓ *such that (a)* $(\mathbf{add}(\ell, *) \to *) \in \mathsf{Qu}_i$ *or* $(\mathbf{add}(*, \ell) \to *) \in \mathsf{Qu}_i$ *and (b)* $\ell \notin \mathsf{Var}(\mathsf{Qu}_i)$. *Let* VQ_i *be the set of all labels* ℓ *such that (a)* $\ell \in \mathsf{Q}_i$ *and (b)* $\ell \in \mathbf{label}(*)$. *Note that* $\mathsf{VQ}_i \subseteq \mathsf{Q}_i$.
2. *Recalling* QEnc *from Construction 7 let* VQ *contain all labels* ℓ *such that (a)* $\ell \in \mathbf{label}(*)$, *(b)* $(\mathbf{add}(\ell, *) \to *) \in \mathsf{QEnc}$ *or* $(\mathbf{add}(*, \ell) \to *) \in \mathsf{QEnc}$, *and (c)* $\ell \notin \mathsf{Var}(\mathsf{QEnc})$. *Note that* VQ *and* VQ_h *are identically distributed.*
3. *Recalling* QGen_i *from Construction 7 let* V_i *contain all labels* ℓ *such that* $\ell \in \mathsf{Var}(\mathsf{QGen}_i)$.

4. *Recalling* QCMP *from Construction 7 we construct* VCMP_i *in an inductive way. First, add to* VCMP_i *all labels* ℓ *such that there exists* $((\ell_1, \ell_2) \xrightarrow[\mathbf{add}]{} \ell) \in \mathsf{QCMP}$ *where* $\ell_1 \in \mathsf{V}_i$ *and* $\ell_2 \in \mathsf{V}_i$*. Then, as long as there exist* $((\ell_1, \ell_2) \xrightarrow[\mathbf{add}]{} \ell) \in \mathsf{QCMP}$ *where* $\ell_1 \in (\mathsf{VCMP}_i \cup \mathsf{V}_i)$ *and* $\ell_2 \in (\mathsf{VCMP}_i \cup \mathsf{V}_i)$*, add* ℓ *to* VCMP_i*.*

5.2 Proof of Correctness

Proposition 3. *Suppose* $n \geq \lceil \frac{2(|\mathsf{pp}|+u)}{3\kappa} + \frac{1}{3} \rceil$ *where* u *is the total number of queries made by* $\mathsf{Enc}(\mathsf{pp}, *)$ *to* $\mathbb{G}_{RR}$*. For random variables as in Definition 12*

$$\Pr[\exists i : ((\mathsf{V}_i \cup \mathsf{VCMP}_i) \cap \mathsf{VQ}_i) \subseteq \cup_{j \neq i}(\mathsf{V}_j \cup \mathsf{VCMP}_j)] \geq 1 - 2^{-\kappa/2}.$$

Proof. We use notation from Definition 12. Let $t = |\cup_{i=1}^{n} \mathsf{Q}_i|$ be the number of distinct labels in $\cup_{i=1}^{n}\mathsf{Q}_i$ and let $w = \lceil \frac{2(|\mathsf{pp}|+u)}{3\kappa} + \frac{1}{3} \rceil$. Let $q = |\cup_{i=1}^{n} \mathsf{VQ}_i|$ be the number of distinct labels in $\cup_{i=1}^{n}\mathsf{VQ}_i$ and recall all labels in $\cup_{i=1}^{n}\mathsf{VQ}_i$ are valid. Since for $i \in [n]$, $\mathsf{VQ}_i \subseteq \mathsf{Q}_i$, then $\cup_{i=1}^{n}\mathsf{VQ}_i \subseteq \cup_{i=1}^{n}\mathsf{Q}_i$.
Based on Lemma 6, $\Pr[q \geq w] \leq 2^{-\kappa/2}$. To see why the former claim holds, view pp in Construction 7 as z in Lemma 6 and view $\cup_{i=1}^{n}\mathsf{Q}_i$ as Chal. Let Small be the event that $q < n$. Since $n \geq \lceil \frac{2(|\mathsf{pp}|+u)}{3\kappa} + \frac{1}{3} \rceil$, $\Pr[\mathsf{Small}] \geq 1 - 2^{-\kappa/2}$.

We now show assuming Small holds, there exists an index $i \in [n]$ such that $((\mathsf{V}_i \cup \mathsf{VCMP}_i) \cap \mathsf{VQ}_i) \subseteq \cup_{j \neq i}(\mathsf{V}_j \cup \mathsf{VCMP}_j)$. Suppose not. Then, for all $k \in [n]$. there exists a label ℓ_k such that $\ell_k \in ((\mathsf{V}_k \cup \mathsf{VCMP}_k) \cap \mathsf{VQ}_k)$ but $\ell_k \notin \cup_{i \neq k}(\mathsf{V}_i \cup \mathsf{VCMP}_i)$. We claim $\ell_1, \ldots, \ell_n$ are distinct labels, and since for all i, $\ell_i \in \mathsf{VQ}_i$, the event $\overline{\mathsf{Small}}$ holds, a contradiction. If for two different indices $j, k \in [n]$, $\ell_j = \ell_k$, then $\ell_k \in \mathsf{V}_j \cup \mathsf{VCMP}_j$ since $\ell_j \in (\mathsf{V}_j \cup \mathsf{VCMP}_j) \cap \mathsf{VQ}_j$. This contradicts the assumption that $\ell_k \notin \cup_{i \neq k}(\mathsf{V}_i \cup \mathsf{VCMP}_i)$. The bound of the lemma now follows. □

Proposition 4. *Let* $\ell \leftarrow \mathcal{A}^{\mathbb{G}_{RR}}(1^\kappa)$ *be an arbitrary poly-query algorithm that takes in the security parameter* 1^κ*, and outputs a label* ℓ*. Let* Q' *be the set of all labels generated by calling* $\mathbb{G}_{RR}$ *during* $\mathcal{A}$*'s execution. Then,*

$$\Pr[\ell \notin \mathsf{Q}' \wedge \ell \in \mathbf{label}(*)] \leq 2^{-2\kappa}.$$

Proof. Suppose that $\mathcal{A}$ checks r public keys using the oracle before outputting the final guess. Recalling the definition of **add** from 6, note that $\mathcal{A}$ can check validity of a label ℓ_1 by querying $\mathbf{add}(\ell_1, \ell_1)$ and if the response was not $\perp$, it can conclude that ℓ' is valid. Since $\mathcal{A}$ is a poly-query algorithm, $q = |\mathsf{Q}'|+r = \mathsf{poly}(\kappa)$. Since $\mathbf{label} : \mathbb{Z}_p \mapsto S$ and $p = 2^\kappa$, there are at most 2^κ valid labels. Let Bad_i for $1 \leq i \leq t$ be the event that the ith label that is checked by the adversary is not valid. Then, the probability that $\mathcal{A}$'s final guess is valid is:

$$1 - \Pr[\mathsf{Bad}_{r+1}] = 1 - \frac{2^{3\kappa} - 2^\kappa - r}{2^{3\kappa} - q} \prod_{i=1}^{r} \Pr[\mathsf{Bad}_i] < 1 - (\frac{2^{3\kappa} - 2^\kappa - r}{2^{3\kappa} - q})^{r+1}$$

For sufficiently large κ, $\frac{2^{3\kappa}-2^{\kappa}-r}{2^{3\kappa}-q} < 1$. Therefore,

$$1-\Pr[\mathsf{Bad}_{r+1}] < 1-(\frac{2^{3\kappa}-2^{\kappa}-r}{2^{3\kappa}-q})\sum_{i=1}^{r}(\frac{2^{3\kappa}-2^{\kappa}-r}{2^{3\kappa}-q})^{i} < (\frac{2^{\kappa}-|Q'|}{2^{3\kappa}-q})(r+1).$$

Therefore, for sufficiently large κ, $\Pr[\ell \in \mathsf{Q}' \wedge \ell \in \mathbf{label}(*)]$ is bounded by $2^{-2\kappa}$. □

Lemma 7. *Suppose we update* $\mathsf{Known} \leftarrow \mathsf{Upd}(\cup_{i\neq h}\mathsf{QGen}_i \cup \mathsf{QCMP} \cup \mathsf{QEnc})$. *Then, assuming*

$$((\mathsf{V}_h \cup \mathsf{VCMP}_h) \cap \mathsf{VQ}) \subseteq \cup_{i\neq h}(\mathsf{V}_i \cup \mathsf{VCMP}_i), \tag{4}$$

with probability at least $1-\mathsf{poly}(\kappa).2^{-2\kappa}$ *for any label that* $\ell \in \mathsf{Var}(\cup_{i\neq h}\mathsf{QGen}_i \cup \mathsf{QCMP} \cup \mathsf{QEnc})$ *but* $\ell \notin \mathsf{VCMP}_h$, $\mathsf{Known}(\ell) = \top$.

Proof. Suppose there exists a label $\ell \in \mathsf{Var}(\cup_{i\neq h}\mathsf{QGen}_i \cup \mathsf{QCMP} \cup \mathsf{QEnc})$ such that $\ell \notin \mathsf{VCMP}_h$ and $\mathsf{Known}(\ell) = \perp$. There are three cases:

- $\ell \in \mathsf{Var}(\mathsf{QGen}_i)$ where $i \neq h$: Let ℓ be the first label in $\mathsf{Var}(\mathsf{QGen}_i)$ such that $\mathsf{Known}(\ell) = \perp$. $\ell \in \mathsf{Var}(\mathsf{QGen}_i)$; thus, as per Definition 9, there exists a Q-A pair $((\ell_x, \ell_y) \xrightarrow[\mathbf{add}]{} \ell) \in \mathsf{QGen}_i$. Therefore $\mathsf{Known}(\ell_x) = \perp$ and $\mathsf{Known}(\ell_y) = \perp$ because otherwise $\mathsf{Known}(\ell) = \top$. Since ℓ is the first label in $\mathsf{Var}(\mathsf{QGen}_i)$ whose $\mathsf{Known} = \perp$, we must have $\ell_x \notin \mathsf{Var}(\mathsf{QGen}_i) \wedge \ell_y \notin \mathsf{Var}(\mathsf{QGen}_i)$. Moreover, ℓ_x, ℓ_y must be valid because otherwise, $\ell = \perp$ and $\ell \notin \mathsf{Var}(\mathsf{QGen}_i)$. Based on Proposition 4, the probability that $\ell_x \notin \mathsf{Var}(\mathsf{QGen}_i) \wedge \ell_y \notin \mathsf{Var}(\mathsf{QGen}_i)$ but ℓ_x, ℓ_y are valid is at most $2^{-4\kappa}$.
- $\ell \in \mathsf{Var}(\mathsf{QCMP}) \wedge \ell \notin \mathsf{VCMP}_h$: Let ℓ be the first label in $\mathsf{Var}(\mathsf{QCMP})$ such that $\mathsf{Known}(\ell) = \perp$ and $\ell \notin \mathsf{VCMP}_h$. $\ell \in \mathsf{Var}(\mathsf{QCMP})$; thus, as per Definition 9, there exists a Q-A pair $((\ell_x, \ell_y) \xrightarrow[\mathbf{add}]{} \ell) \in \mathsf{QCMP}$. Therefore $\mathsf{Known}(\ell_x) = \perp$ and $\mathsf{Known}(\ell_y) = \perp$ because otherwise $\mathsf{Known}(\ell) = \top$. Thus, ℓ_x and ℓ_y cannot be in $\cup_{i\neq h}\mathsf{Var}(\mathsf{QGen}_i)$ because otherwise they will be known with probability $1-2^{-4\kappa}$ based on the previous case. Since ℓ is the first label in $\mathsf{Var}(\mathsf{QCMP})$ whose $\mathsf{Known} = \perp$ and is not in VCMP_h, we must have ($\ell_x \notin \mathsf{Var}(\mathsf{QCMP})$ or $\ell_x \in \mathsf{VCMP}_h$) and ($\ell_y \notin \mathsf{Var}(\mathsf{QCMP})$ or $\ell_y \in \mathsf{VCMP}_h$). Moreover, ℓ_x, ℓ_y must be valid because otherwise, $\ell = \perp$ and $\ell \notin \mathsf{Var}(\mathsf{QCMP})$. If both $\ell_x, \ell_y \in \mathsf{V}_h \cup \mathsf{VCMP}_h$, then $\ell \in \mathsf{VCMP}_h$. Therefore, at most one of ℓ_x, ℓ_y can be in $\mathsf{V}_h \cup \mathsf{VCMP}_h$ and at least one of ℓ_x, ℓ_y must not be in $\mathsf{Var}(\mathsf{QCMP} \cup \mathsf{QGen}_h)$. Thus, at least one of ℓ_x, ℓ_y must not be in $\mathsf{Var}(\mathsf{QCMP} \cup_{i=1}^{n} \mathsf{QGen}_i)$. Based on Proposition 4, the probability that this case happens is at most $2^{-2\kappa} + 2^{-4\kappa}$ which is bounded by $2^{-\kappa+1}$.
- $\ell \in \mathsf{Var}(\mathsf{QEnc})$: Let ℓ be the first label in $\mathsf{Var}(\mathsf{QEnc})$ such that $\mathsf{Known}(\ell) = \perp$. $\ell \in \mathsf{Var}(\mathsf{QEnc})$; thus, as per Definition 9, there exists a Q-A pair $((\ell_x, \ell_y) \xrightarrow[\mathbf{add}]{} \ell) \in \mathsf{QEnc}$. Therefore $\mathsf{Known}(\ell_x) = \perp$ and $\mathsf{Known}(\ell_y) = \perp$ because otherwise $\mathsf{Known}(\ell) = \top$. Since ℓ is the first label in $\mathsf{Var}(\mathsf{QEnc})$ whose $\mathsf{Known} = \perp$, we

must have $\ell_x \notin \mathsf{Var}(\mathsf{QEnc}) \wedge \ell_y \notin \mathsf{Var}(\mathsf{QEnc})$. Moreover, ℓ_x, ℓ_y must be valid because otherwise, $\ell = \bot$ and $\ell \notin \mathsf{Var}(\mathsf{QEnc})$. Based on 4, if $\ell_x \in \mathsf{V}_h \cup \mathsf{VCMP}_h$, then $\ell_x \in \cup_{i \neq h} \mathsf{V}_i \cup \mathsf{VCMP}_i$ and based on the two previous cases, $\mathsf{Known}(\ell_x) = \top$ with probability $1 - 2^{-2\kappa}$. Therefore, $\ell_x \notin \mathsf{V}_h \cup \mathsf{VCMP}_h$. The same argument holds for ℓ_y. Thus, both of ℓ_x, ℓ_y are not in $\mathsf{Var}(\mathsf{QCMP} \cup_{i=1}^{n} \mathsf{QGen}_i \cup \mathsf{QEnc})$. Thus, based on Proposition 4, the probability that ℓ_x, ℓ_y are valid labels is at most $2^{-4\kappa}$. Therefore, the probability that this case happens is at most $2^{-2\kappa+1}$.

Therefore, with probability at least $p_1 = 1 - \mathsf{poly}(\kappa).2^{-2\kappa}$, for any label that $\ell \in \mathsf{Var}(\cup_{i \neq h} \mathsf{QGen}_i \cup \mathsf{QCMP} \cup \mathsf{QEnc})$ but $\ell \notin \mathsf{VCMP}_h$, $\mathsf{Known}(\ell) = \top$.

□

Proposition 5. *Assuming*

$$((\mathsf{V}_h \cup \mathsf{VCMP}_h) \cap \mathsf{VQ}) \subseteq \cup_{i \neq h} (\mathsf{V}_i \cup \mathsf{VCMP}_i), \tag{5}$$

the probability that the algorithm Ver *(Construction 7) doesn't output the correct bit is at most* $1 - 2^{-2\kappa/3}$.

Definition 13 (Inconsistency). *Suppose* Se *is a set of Q-A pairs. We say a Q-A pair* $((\ell_1, \ell_2) \xrightarrow[\mathbf{add}]{} \ell')$ *is inconsistent with* Se *if one the following happens:*

- $\mathsf{Inconsistent}_1$*: This case happens if Q-A pairs in* Se *imply that* $((\ell_1, \ell_2) \xrightarrow[\mathbf{add}]{} \bot)$ *while* $\ell' \neq \bot$*. In other words,* $\mathsf{Inconsistent}_1$ *happens if:*
  1. $\ell_1 \notin \mathsf{Var}(\mathsf{Se})$ *or* $\ell_2 \notin \mathsf{Var}(\mathsf{Se})$*. Let* $b \in \{1, 2\}$ *be an index where* $\ell_b \notin \mathsf{Var}(\mathsf{Se})$.
  2. *There exists a Q-A pair* $((\ell_b, \widetilde{\ell}) \xrightarrow[\mathbf{add}]{} \bot) \in \mathsf{Se}$ *or* $((\widetilde{\ell}, \ell_b) \xrightarrow[\mathbf{add}]{} \bot) \in \mathsf{Se}$ *where* $\widetilde{\ell} \in \mathsf{Var}(\mathsf{Se})$ *or* $\widetilde{\ell} = \ell_j$ *where* $j = 1$ *or* $j = 2$.
  3. $\ell' \neq \bot$.
- $\mathsf{Inconsistent}_2$*: This case happens if Q-A pairs in* Se *imply that* $((\ell_1, \ell_2) \xrightarrow[\mathbf{add}]{} \ell^* \neq \bot)$ *while* $\ell' \neq \ell^*$*. In other words,* $\mathsf{Inconsistent}_2$ *happens if:*
  1. $x_{\ell_1} + x_{\ell_2} - x_{\ell^*} \in \mathsf{Span}(\mathsf{Eq}(\mathsf{Se}))$.
  2. $\ell' \neq \ell^*$.

Claims for Correctness. Suppose $\mathsf{qu} = \mathbf{add}(\ell_1, \ell_2)$ is the first query inside DecSim whose response creates an inconsistency as per Definition 13. Let ℓ' be the response generated by DecSim and view $(\cup \mathsf{QGen}_i \cup \mathsf{QCMP} \cup \mathsf{QEnc} \cup \mathsf{T})$ where T is the previous set of Q-A pairs in DecSim as Se in Definition 13.

- If qu results in $\mathsf{Inconsistent}_1$, then we must have $\ell_j \notin \mathsf{Var}(\cup \mathsf{QGen}_i \cup \mathsf{QCMP} \cup \mathsf{QEnc} \cup \mathsf{T})$ where $j = 1$ or $j = 2$.
  we must also have $\ell' \neq \bot$ because otherwise we would not run into an inconsistency. Considering the DecSim procedure in answering queries, $\ell' \neq \bot$ if and only if $\ell_1 \in \mathsf{V} \cup \mathsf{Var}(\mathsf{QGen}_h)$ and $\ell_2 \in \mathsf{V} \cup \mathsf{Var}(\mathsf{QGen}_h)$. It is easy to see that $\mathsf{V} \cup \mathsf{Var}(\mathsf{QGen}_h) = \mathsf{Var}(\cup \mathsf{QGen}_i \cup \mathsf{QCMP} \cup \mathsf{QEnc} \cup \mathsf{T})$ which contradicts $\ell_j \in (\mathsf{V} \cup \mathsf{Var}(\mathsf{QGen}_h))$ for $j = 1, 2$. Therefore, $\mathsf{Inconsistent}_1$ does not happen.

– If qu results in $\mathsf{Inconsistent}_2$, then:
  1. $x_{\ell_1} + x_{\ell_2} - x_{\ell^*}$ must be in $\mathsf{Span}(\mathsf{Eq}(\cup\mathsf{QGen}_i \cup \mathsf{QCMP} \cup \mathsf{QEnc} \cup \mathsf{T}))$.
  2. $\ell' \neq \ell^*$.

  First we argue that in order for $\mathsf{Inconsistent}_2$ to occur, we must have $\ell^* \in \mathsf{Var}(\cup\mathsf{QGen}_\mathsf{i} \cup \mathsf{QCMP} \cup \mathsf{QEnc} \cup \mathsf{T})$. Let's call this condition Cond. Suppose not. Then, there is no Q-A pair such that

$$((*,*) \xrightarrow[\mathbf{add}]{} \ell^*) \in \cup\mathsf{QGen}_\mathsf{i} \cup \mathsf{QCMP} \cup \mathsf{QEnc} \cup \mathsf{T}.$$

  Thus, there must be a query $((\ell^*,*) \xrightarrow[\mathbf{add}]{} l_y) \in \cup\mathsf{QGen}_\mathsf{i} \cup \mathsf{QCMP} \cup \mathsf{QEnc} \cup \mathsf{T}$ or $((*,\ell^*) \xrightarrow[\mathbf{add}]{} l_y) \in \cup\mathsf{QGen}_\mathsf{i} \cup \mathsf{QCMP} \cup \mathsf{QEnc} \cup \mathsf{T}$ because otherwise $x_{\ell_1} + x_{\ell_2} - x_{\ell^*}$ could have not been in $\mathsf{Span}(\mathsf{Eq}(\cup\mathsf{QGen}_i \cup \mathsf{QCMP} \cup \mathsf{QEnc} \cup \mathsf{T}))$. Since $\ell^* \notin \mathsf{Var}(\cup\mathsf{QGen}_\mathsf{i} \cup \mathsf{QCMP} \cup \mathsf{QEnc} \cup \mathsf{T})$, Proposition 4 implies that ℓ^* is invalid with probability $1 - 2^{-2\kappa}$. Therefore, $l_y = \perp$ with probability $1 - 2^{-2\kappa}$. Since equations containing $\perp$ will not be added to the $\mathsf{Eq}(*)$, no equation containing x_{ℓ^*} will be added to $\mathsf{Eq}(\cup\mathsf{QGen}_i \cup \mathsf{QCMP} \cup \mathsf{QEnc} \cup \mathsf{T})$ which contradicts $x_{\ell_1} + x_{\ell_2} - x_{\ell^*} \in \mathsf{Span}(\mathsf{Eq}(\cup\mathsf{QGen}_i \cup \mathsf{QCMP} \cup \mathsf{QEnc} \cup \mathsf{T}))$. Therefore, if Cond does not hold, the probability that qu causes $\mathsf{Inconsistent}_2$ is at most $2^{-2\kappa}$.

  Now, suppose there exist a Q-A pair $((\ell'_1,\ell'_2) \xrightarrow[\mathbf{add}]{} \ell^*) \in \cup\mathsf{QGen}_i \cup \mathsf{QCMP} \cup \mathsf{QEnc} \cup \mathsf{T}$. We consider all possible cases:

  1. $\ell^* \in \mathsf{Var}(\mathsf{QEnc})$: Suppose $((\ell'_1,\ell'_2) \xrightarrow[\mathbf{add}]{} \ell^*) \in \mathsf{QEnc}$. If both $\ell_1, \ell_2 \in \mathsf{V}$, then considering Condition 1, by Item 2b of DecSim, $\ell' = \ell^*$. Else, we claim that $\mathsf{Known}(\ell^*) = \top$; thus, considering Condition 1, by Item 2d of DecSim, $\ell' = \ell^*$. If $\mathsf{Known}(\ell^*) = \perp$, then $\mathsf{Known}(\ell'_1) = \perp$ and $\mathsf{Known}(\ell'_2) = \perp$. Recall updating Known in 2 and 3. Based on Lemma 7, if $\ell'_1 \in \mathsf{Var}(\cup_{i\neq h}\mathsf{QGen}_i \cup \mathsf{QCMP} \cup \mathsf{QEnc})\backslash\mathsf{VCMP}_h$, with probability p_1, we must have $\mathsf{Known}(\ell'_1) = \top$. Suppose $\ell'_1 \notin \mathsf{Var}(\cup_{i\neq h}\mathsf{QGen}_i \cup \mathsf{QCMP} \cup \mathsf{QEnc})\backslash\mathsf{VCMP}_h$, thus there are two possible cases:
     (a) $\ell'_1 \in \mathsf{Var}(\mathsf{QGen}_h) \cup \mathsf{VCMP}_h$: Considering Definition 12, $\ell'_1 \in (\mathsf{V}_h \cup \mathsf{VCMP}_h) \cap \mathsf{VQ}$ but $\ell'_1 \notin \cup_{i\neq h}(\mathsf{V}_i \cup \mathsf{VCMP}_i)$ which contradicts Equation 5.
     (b) $\ell'_1 \notin \mathsf{Var}(\mathsf{All} \cup \mathsf{QGen}_h)$: ℓ'_1 must be valid because otherwise $\ell^* = \perp$ and qu could not have caused $\mathsf{Inconsistent}_2$. Thus, by Proposition 4, the probability of this case happening is at most $2^{-2\kappa}$.

     Therefore, the probability that $\mathsf{Known}(\ell'_1) = \perp$ and $\mathsf{Known}(\ell'_2) = \perp$ is at most $(1 - p_1 + 2^{-2\kappa})^2 = \mathsf{poly}(\kappa).2^{-4\kappa}$. Thus, the probability that $\ell^* \in \mathsf{Var}(\mathsf{QEnc})$ and $\mathsf{Known}(\ell^*) = \perp$ is bounded by $2^{-3\kappa}$.
  2. $\ell^* \in \mathsf{Var}(\cup_i\mathsf{QGen}_i \cup \mathsf{QCMP})$: Suppose $((\ell'_1,\ell'_2) \xrightarrow[\mathbf{add}]{} \ell^*) \in \cup_i\mathsf{QGen}_i \cup \mathsf{QCMP}$. If $\ell_1 \in \mathsf{V} \wedge \ell_2 \in \mathsf{V}$, considering Condition 1 above, Item 2b of DecSim, implies that $\ell' = \ell^*$. Else, if $\ell_1 \in \mathsf{Var}(\mathsf{QGen}_h)$ or $\ell_2 \in \mathsf{Var}(\mathsf{QGen}_h)$, considering Condition 1 above, Item 2c of DecSim, implies that $\ell' = \ell^*$.

3. $\ell^* \in \mathsf{Var}(\mathsf{T})$: Since $\ell^* \notin \mathsf{Var}(\mathsf{All} \cup \mathsf{QGen}_\mathsf{h})$, ℓ^* must be one of the labels that are randomly generated by DecSim. Therefore, based on the definition of DecSim, $\mathsf{Known}(\ell^*) = \top$. Thus, if $\ell_1 \in \mathsf{V} \wedge \ell_2 \in \mathsf{V}$, considering Condition 1 above, Item 2b of DecSim, implies that $\ell' = \ell^*$ and if $\ell_1 \in \mathsf{Var}(\mathsf{QGen}_h)$ or $\ell_2 \in \mathsf{Var}(\mathsf{QGen}_h)$, considering Condition 1 above, Item 2c of DecSim, implies that $\ell' = \ell^*$.

Thus, if Cond holds, the probability that qu causes $\mathsf{Inconsistent}_2$, is at most $2^{-3\kappa}$. Therefore, Ver doesn't output the correct bit with probability at most $\mathsf{poly}(\kappa).(2^{-3\kappa} + 2^{-2\kappa}) \leq 2^{-2\kappa/3}$.

Lemma 8. *Suppose Π is the signature scheme defined in Construction 7 with oracle access of the form $(\mathrm{Gen}^{\mathbb{G}_{RR}}, \mathrm{Sig}, \mathrm{Ver}^{\mathbb{G}_{RR}})$ and the PKCom scheme underlying Π is $(1-\epsilon)$-correct. Then, Π is $(1-\epsilon)\frac{(1-2^{-\kappa/3})}{n}$ correct.*

Proof. Let $\mathsf{Success}$ be the event that the verification algorithm outputs the correct bit. In other words, if $\mathsf{Success}_i$ holds, we have: $\mathrm{Ver}^{\mathbb{G}_{RR}}(\mathsf{vrk}, i, \mathrm{Sig}(\mathsf{sgk}, i)) = 1$ where $(\mathsf{sgk}, \mathsf{vrk}) \leftarrow \mathrm{Gen}^{\mathbb{G}_{RR}}(1^\kappa, i)$. Finally, let X be the random variable denoting the message chosen to be signed. Also, let Good be the event that there exists $h \in [n]$ for which Proposition 3 holds. In other words, if Good happens, we have:

$$((\mathsf{V}_h \cup \mathsf{VCMP}_h) \cap \mathsf{VQ}) \subseteq \cup_{j \neq h}(\mathsf{V}_j \cup \mathsf{VCMP}_j)$$

Based on Proposition 3, $\Pr[\mathsf{Good}] \geq 1 - 2^{-\kappa/2}$. Therefore:

$$\Pr[\Pi \text{ is correct}] = \sum_{i=1}^{n} \Pr[\mathsf{Success}|\mathsf{X} = i] \Pr[\mathsf{X} = i] = \frac{1}{n} \sum_{i=1}^{n} \Pr[\mathsf{Success}_i]$$

$$\Pr[\mathsf{Success}_i] = \Pr[\mathsf{Success}_i|\mathsf{Good}] \Pr[\mathsf{Good}] + \Pr[\mathsf{Success}_i|\overline{\mathsf{Good}}] \Pr[\overline{\mathsf{Good}}] \geq$$
$$\Pr[\mathsf{Success}_i|\mathsf{Good}] \Pr[\mathsf{Good}] \geq (1 - 2^{-\kappa/2}) \Pr[\mathsf{Success}_i|\mathsf{Good}]$$

$$\Rightarrow \Pr[\Pi \text{ is correct}] = \frac{1}{n} \sum_{i=1}^{n} \Pr[\mathsf{Success}_i] \geq \frac{1}{n}(1 - 2^{-\kappa/2}) \sum_{i=1}^{n} \Pr[\mathsf{Success}_i|\mathsf{Good}] \geq$$

$$\frac{1}{n}(1 - 2^{-\kappa/2}) \Pr[\mathsf{Success}_h|\mathsf{Good}] \geq \frac{1}{n}(1 - 2^{-\kappa/2})(1 - 2^{-2\kappa/3}) \geq \frac{(1 - 2^{-\kappa/3})}{n}$$

□

5.2.1 Security

Lemma 9 (Security of Construction 7). *Construction 7 is one-time unforgeable if the PKCom scheme is secure.*

Proof. Our proof of security follows exactly the same steps made in Lemma 5. Namely, the adversary against the PKCMP uses its full oracle access to **add** to make up for its lack of access to QEnc.

We show how to transform an adversary $\mathcal{B}$ against the signatures into an adversary $\mathcal{A}^{\mathcal{B}, \mathbb{G}_{RR}}$ against PKCom.

1. $\mathcal{A}$ samples h, and for $i \in [n] \setminus \{h\}$, it samples $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{Key}^{\mathbb{G}_{RR}}(1^\kappa)$ and collects the Q-A pairs made to $\mathbb{G}_{RR}$ in QGen_i. It then gives $(\mathsf{pk}_i, \mathsf{sk}_i)_{i \neq h}$ to the challenger to receive $(\mathsf{pk}_h, \mathsf{ct})$.
2. $\mathcal{A}$ runs $\mathsf{Com}^{\mathbb{G}_{RR}}(\mathsf{pk}_1, \ldots, \mathsf{pk}_n)$ and adds all Q-A pairs made to $\mathbb{G}_{RR}$ to QCMP. Then, $\mathcal{A}$ updates $\mathsf{Known} \leftarrow \mathsf{Upd}(\cup_{i \neq h} \mathsf{QGen}_i \cup \mathsf{QCMP})$ forms $\mathsf{vrk} := (\{\mathsf{pk}_i\}, \cup_{j \neq h} \mathsf{QGen}_j \cup \mathsf{QCMP}, \mathsf{Known}, \upsilon)$, where $\upsilon = \mathbf{label}(1)$.
3. $\mathcal{A}$ runs $(\mathsf{sk}'_h, \mathsf{QGen}'_h) \leftarrow \mathcal{B}(\mathsf{vrk}, h)$.
4. $\mathcal{A}$ runs $(\mathsf{ct}', *) \leftarrow \mathsf{Enc}^{\mathbb{G}_{RR}}(\mathsf{pp}, h)$ η times, each time recording the queries in a set QEnc', and then runs DecSim while using QEnc', records all queries in $\mathsf{collect}$ and updates $\mathsf{Known} \leftarrow \mathsf{Upd}(\mathsf{collect})$.
5. $\mathcal{A}$ runs $\mathsf{DecSim}_0(\mathsf{sk}'_h, h, \mathsf{ct})$: Let $\mathsf{All}' = \cup_{j \neq h} \mathsf{QGen}_j \cup \mathsf{QCMP} \cup \mathsf{collect}$ and let $\mathsf{E} = \mathsf{Eq}(\mathsf{All}')$ and $\mathsf{V} = \mathsf{Var}(\mathsf{All}')$. For a query $\mathsf{qu} := \mathbf{add}(\ell_1, \ell_2)$:
   (a) if $\ell_1 \notin \mathsf{V} \cup \mathsf{Var}(\mathsf{QGen}'_h)$ or $\ell_2 \notin \mathsf{V} \cup \mathsf{Var}(\mathsf{QGen}'_h)$, return $\perp$.
   (b) if $\ell_1 \in \mathsf{V}$ and $\ell_2 \in \mathsf{V}$ if there exists $\ell \in \mathsf{V} \cup \mathsf{Var}(\mathsf{QGen}'_h)$ such that $x_{\ell_1} + x_{\ell_2} - x_\ell \in \mathsf{Span}(\mathsf{E} \cup \mathsf{Eq}(\mathsf{QGen}_h))$, return ℓ. If no such an ℓ is found, respond with a random label ℓ', add $x_{\ell_1} + x_{\ell_2} - x_{\ell'}$ to E and add ℓ' to V. Also, set $\mathsf{Known}(\ell') = \top$.
   (c) Else if there exists a label ℓ such that $x_{\ell_1} + x_{\ell_2} - x_{\ell'} \in \mathsf{Span}(\mathsf{Eq}(\mathsf{QCMP} \cup_i \mathsf{QGen}_i))$, return ℓ;
   (d) Else, if there exists a label ℓ such that $\mathsf{Known}(\ell) = \top$ and $x_{\ell_1} + x_{\ell_2} - x_\ell \in \mathsf{Span}(\mathsf{E} \cup \mathsf{Eq}(\mathsf{QGen}_h))$, return ℓ. Else, respond with a random label ℓ', add $x_{\ell_1} + x_{\ell_2} - x_{\ell'}$ to E, and add ℓ' to V. Also, set $\mathsf{Known}(\ell') = \top$.
   (e) Else, respond to qu with $\mathbf{add}(\ell_1, \ell_2)$.

□

References

BB04. Dan Boneh and Xavier Boyen. Secure identity based encryption without random oracles. In Matthew Franklin, editor, *Advances in Cryptology – CRYPTO 2004*, volume 3152 of *Lecture Notes in Computer Science*, pages 443–459, Santa Barbara, CA, USA, August 15–19, 2004. Springer, Heidelberg, Germany. 3

BF01. Dan Boneh and Matt Franklin. Identity-based encryption from the weil pairing. In *Advances in Cryptology—CRYPTO 2001: 21st Annual International Cryptology Conference, Santa Barbara, California, USA, August 19–23, 2001 Proceedings*, pages 213–229. Springer, 2001. 1, 2, 3

BLSV18. Zvika Brakerski, Alex Lombardi, Gil Segev, and Vinod Vaikuntanathan. Anonymous ibe, leakage resilience and circular security from new assumptions. In Jesper Buus Nielsen and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2018*, pages 535–564, Cham, 2018. Springer International Publishing. 3

BPR+08. Dan Boneh, Periklis A. Papakonstantinou, Charles Rackoff, Yevgeniy Vahlis, and Brent Waters. On the impossibility of basing identity based encryption on trapdoor permutations. In *49th Annual Symposium on Foundations of Computer Science*, pages 283–292, Philadelphia, PA, USA, October 25–28, 2008. IEEE Computer Society Press. 3, 4, 5, 7

CFGG23. Dario Catalano, Dario Fiore, Rosario Gennaro, and Emanuele Giunta. On the impossibility of algebraic vector commitments in pairing-free groups. In *Theory of Cryptography: 20th International Conference, TCC 2022, Chicago, IL, USA, November 7–10, 2022, Proceedings, Part II*, pages 274–299. Springer, 2023. 6

DG17. Nico Döttling and Sanjam Garg. From selective IBE to full IBE and selective HIBE. 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 372–408, Baltimore, MD, USA, November 12–15, 2017. Springer, Heidelberg, Germany. 3

DGI+19. Nico Döttling, Sanjam Garg, Yuval Ishai, Giulio Malavolta, Tamer Mour, and Rafail Ostrovsky. Trapdoor hash functions and their applications. In Alexandra Boldyreva and Daniele Micciancio, editors, *Advances in Cryptology – CRYPTO 2019, Part III*, volume 11694 of *Lecture Notes in Computer Science*, pages 3–32, Santa Barbara, CA, USA, August 18–22, 2019. Springer, Heidelberg, Germany. 5

DHH+21. Nico Döttling, Dominik Hartmann, Dennis Hofheinz, Eike Kiltz, Sven Schäge, and Bogdan Ursu. On the impossibility of purely algebraic signatures. In Kobbi Nissim and Brent Waters, editors, *TCC 2021: 19th Theory of Cryptography Conference, Part III*, volume 13044 of *Lecture Notes in Computer Science*, pages 317–349, Raleigh, NC, USA, November 8–11, 2021. Springer, Heidelberg, Germany. 6

DP23. Pratish Datta and Tapas Pal. Registration-based functional encryption. Cryptology ePrint Archive, Paper 2023/457, 2023. 2

FFM+23. Danilo Francati, Daniele Friolo, Monosij Maitra, Giulio Malavolta, Ahmadreza Rahimi, and Daniele Venturi. Registered (inner-product) functional encryption. Cryptology ePrint Archive, Paper 2023/395, 2023. 2

GHM+19. Sanjam Garg, Mohammad Hajiabadi, Mohammad Mahmoody, Ahmadreza Rahimi, and Sruthi Sekar. Registration-based encryption from standard assumptions. In Dongdai Lin and Kazue Sako, editors, *Public-Key Cryptography – PKC 2019*, pages 63–93, Cham, 2019. Springer International Publishing. 1, 2

GHMR18. Sanjam Garg, Mohammad Hajiabadi, Mohammad Mahmoody, and Ahmadreza Rahimi. Registration-based encryption: Removing private-key generator from IBE. 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 689–718, Panaji, India, November 11–14, 2018. Springer, Heidelberg, Germany. 1, 2, 13

GKMR22. Noemi Glaeser, Dimitris Kolonelos, Giulio Malavolta, and Ahmadreza Rahimi. Efficient registration-based encryption. Cryptology ePrint Archive, Paper 2022/1505, 2022. https://eprint.iacr.org/2022/1505. 1, 3, 6

GT00. Rosario Gennaro and Luca Trevisan. Lower bounds on the efficiency of generic cryptographic constructions. In *41st Annual Symposium on Foundations of Computer Science*, pages 305–313, Redondo Beach, CA, USA, November 12–14, 2000. IEEE Computer Society Press. 10, 12

GV20. Rishab Goyal and Satyanarayana Vusirikala. Verifiable registration-based encryption. In *Advances in Cryptology – CRYPTO 2020: 40th Annual International Cryptology Conference, CRYPTO 2020, Santa Barbara, CA, USA, August 17–21, 2020, Proceedings, Part I*, page 621–651, Berlin, Heidelberg, 2020. Springer-Verlag. 2

HHRS07. Iftach Haitner, Jonathan J. Hoch, Omer Reingold, and Gil Segev. Finding collisions in interactive protocols - a tight lower bound on the round complexity of statistically-hiding commitments. In *48th Annual Symposium on Foundations of Computer Science*, pages 669–679, Providence, RI, USA, October 20–23, 2007. IEEE Computer Society Press. 10

HLWW23. Susan Hohenberger, George Lu, Brent Waters, and David J. Wu. Registered attribute-based encryption. In Carmit Hazay and Martijn Stam, editors, *Advances in Cryptology – EUROCRYPT 2023*, pages 511–542, Cham, 2023. Springer Nature Switzerland. 1, 2, 3, 4, 6, 13

Mau05. Ueli M. Maurer. Abstract models of computation in cryptography (invited paper). In Nigel P. Smart, editor, *10th IMA International Conference on Cryptography and Coding*, volume 3796 of *Lecture Notes in Computer Science*, pages 1–12, Cirencester, UK, December 19–21, 2005. Springer, Heidelberg, Germany. 3, 4, 5

PRV12. Periklis A. Papakonstantinou, Charles W. Rackoff, and Yevgeniy Vahlis. How powerful are the DDH hard groups? Cryptology ePrint Archive, Report 2012/653, 2012. https://eprint.iacr.org/2012/653. 4, 7

RSS20. Lior Rotem, Gil Segev, and Ido Shahaf. Generic-group delay functions require hidden-order groups. In Anne Canteaut and Yuval Ishai, editors, *Advances in Cryptology – EUROCRYPT 2020, Part III*, volume 12107 of *Lecture Notes in Computer Science*, pages 155–180, Zagreb, Croatia, May 10–14, 2020. Springer, Heidelberg, Germany. 5

RTV04. Omer Reingold, Luca Trevisan, and Salil P. Vadhan. Notions of reducibility between cryptographic primitives. In Moni Naor, editor, *TCC 2004: 1st Theory of Cryptography Conference*, volume 2951 of *Lecture Notes in Computer Science*, pages 1–20, Cambridge, MA, USA, February 19–21, 2004. Springer, Heidelberg, Germany. 4

SGS21. Gili Schul-Ganz and Gil Segev. Generic-group identity-based encryption: A tight impossibility result. In *2nd Conference on Information-Theoretic Cryptography (ITC 2021)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2021. 4, 6, 7

Sho97. Victor Shoup. Lower bounds for discrete logarithms and related problems. In Walter Fumy, editor, *Advances in Cryptology – EUROCRYPT'97*, volume 1233 of *Lecture Notes in Computer Science*, pages 256–266, Konstanz, Germany, May 11–15, 1997. Springer, Heidelberg, Germany. 3, 4, 5, 6, 27

WLW+23. Qin Wang, Rujia Li, Qi Wang, David Galindo, Shiping Chen, and Yang Xiang. Transparent registration-based encryption through blockchain. *Distrib. Ledger Technol.*, 2(1), mar 2023. 2

Zha22. Mark Zhandry. To label, or not to label (in generic groups). In *Advances in Cryptology – CRYPTO 2022, Part III*, Lecture Notes in Computer Science, pages 66–96, Santa Barbara, CA, USA, August 2022. Springer, Heidelberg, Germany. 4, 5, 6, 12, 15, 16, 37

A Omitted Proofs

For sake of completeness, here we include a full of Lemma 1, which is heavily based on that of [Zha22] and is simply adapted to our setting.

Proof (of Lemma 1 - adapted from [Zha22]). Consider choosing an oracle O, a random m, and $(\mathsf{sgk}, \mathsf{vrk}) \leftarrow \mathrm{Gen}^{O}(1^{\kappa}, m)$, and then fixing them. We will say

that σ is "good" if $\Pr[\mathrm{Ver}^O(\mathsf{vrk}, m, \sigma) = 1] \geq \delta/2$, where the probability is taken over the random coins of Ver. By correctness, with probability at least $\delta/2$ over $m, (\mathsf{sgk}, \mathsf{vrk}) \leftarrow \mathrm{Gen}^O(1^\kappa, m)$, there will exist at least one good σ, namely the output of $\sigma \leftarrow \mathrm{Sig}^O(\mathsf{sgk}, \mathsf{m})$.

Suppose Ver0 was deterministic. Then we could compute $v \leftarrow \mathrm{Ver0}^O(\mathsf{vrk}, \mathsf{m})$, and consider the oracle-free probabilistic circuit $C(\sigma) = \mathrm{Ver1}(v, \sigma)$. Then an input σ is good if and only if $C(\sigma)$ accepts with probability at least $\delta/2$. Since C is oracle-free, we can brute-force search for such a σ, finding it with probability at least $\delta/2$. The forgery will then be (m, σ), which is accepted by the challenger with probability $\delta/2$, giving an overall advantage $\delta^2/4$.

For a potentially randomized Ver0, we have to work slightly harder. For a good σ, we have that $\Pr_{v \leftarrow \mathrm{Ver0}^O(\mathsf{vrk},\mathsf{m})}[\Pr[\mathrm{Ver1}(v, \sigma) = 1] \geq \delta/4] \geq \delta/4$. Meanwhile, we will call a σ "bad" if $\Pr_{v \leftarrow \mathrm{Ver0}^O(\mathsf{vrk},\mathsf{m})}[\Pr[\mathrm{Ver1}(v, \sigma) = 1] \geq \delta/4] \leq \delta/8$.

For a parameter t chosen momentarily, we let $v_1, \ldots, v_t \leftarrow \mathrm{Ver0}^O(\mathsf{vrk}, \mathsf{m})$, and construct circuits $C_i(\sigma) = \mathrm{Ver1}(v_i, \sigma)$. We then brute-force search for a σ such that $\Pr_{i \leftarrow [t]}[\Pr[C_i(\sigma) = 1] \geq \delta/4] \geq 3\delta/8$. By Hoeffding's inequality, any good σ will be a solution with probability $1 - 2^{\Omega(\delta^2 t)}$. Meanwhile, any bad σ will be a solution with probability $2^{-\Omega(\delta^2 t)}$. By setting t such that t/δ^2 is sufficiently longer than the bit-length of signatures, we can union bound over all bad δ, showing that there will be no bad solutions except with negligible probability. We will therefore find a not-bad solution with probability at least $\delta/2 - \mathrm{negl} \geq \delta/3$. In this case, with probability at least $\delta/8$ over the choice of v by the verifier, $\Pr[\mathrm{Ver1}(v, \sigma) = 1] \geq \delta/4$. Hence, the overall success probability is at least $(\delta/3) \times (\delta/8) \times (\delta/4) \geq \delta^3/100$. □

We now present proof of Lemma 6.

Proof (Proof of Lemma 6). Let $s = |\mathcal{S}| = 2^{3\kappa}$. Assume wlog that both $\mathcal{A}$ and $\mathcal{B}$ are deterministic. We prove that any fixed labeling function **label** for which Success holds can be uniquely described with

$$f := \log\left(2^{|z|}\binom{p}{w}w!\binom{t}{w}\frac{(s-w)!}{(s-p)!}2^u w!\right) \tag{6}$$

bits.

This means that there exists at most 2^f different Successful oracles. Using the inequalities $(a/b)^b \leq \binom{a}{b} \leq (ae/b)^b$, the fraction of $\mathbf{g}$ oracles for which Success holds is at most

$$\frac{2^f}{\text{number of } L \text{ oracles}} \le \frac{2^{|z|+u} w! \binom{p}{w} w! \binom{t}{w} \frac{(s-w)!}{(s-p)!}}{\frac{s!}{(s-p)!}} = \frac{2^{|z|+u} w! \binom{p}{w} \binom{t}{w}}{\binom{s}{w}}$$
$$\le \frac{2^{|z|+u} w! (\frac{pe}{w})^w (\frac{te}{w})^w}{(\frac{s}{w})^w} = 2^{|z|+u} w! (\frac{e^2 tp}{sw})^w \le 2^{|z|+u} w! (\frac{16 \times 2^{\kappa/3}}{2^{2\kappa} w})^w$$
$$\le 2^{|z|+u} w! (\frac{1}{2^{(3/2)\kappa} w})^w \qquad \text{because } \frac{16 \times 2^{\kappa/3}}{2^{2\kappa}} \le \frac{1}{2^{(3/2)\kappa}} \text{ for large } \kappa$$
$$\le 2^{|z|+u} (\frac{1}{2^{(3/2)\kappa}})^w = \frac{1}{2^{(3/2)\kappa w - |z| - u}} \le \frac{1}{2^{\kappa/2}},$$

as desired. The last inequality follows from $\frac{3}{2}kw - |z| - u \ge k/2$, in turn obtained from $w \ge \frac{2(|z|+u)}{3\kappa} + \frac{1}{3}$.

We now prove Equation 6. Fix a $\mathsf{Successful}$ labelling function **label**. Let $\mathsf{Chal} = \{\ell_1, \ldots, \ell_t\}$ and wlog assume $\ell_1 <_{\mathsf{lex}} \ell_2 <_{\mathsf{lex}} \cdots <_{\mathsf{lex}} \ell_t$, where $\le_{\mathsf{lex}}$ denotes lexicographical ordering. Let $(\ell_{i_1}, \ldots, \ell_{i_w})$ be the w lexicographically smallest elements in Chal that have a pre-image under **label**, and let $(x_{i_1}, \ldots, x_{i_w})$ be their pre-images. Let $\mathsf{Chal}_\mathsf{x} := \{x_{i_1}, \ldots, x_{i_w}\}$.

We say a query to **add** is *new* for $\mathcal{B}$ if it satisfies the following requirements: (1) the answer to this query is not $\perp$; (2) at least one of the input labels has not been input to queries to **add** made by $\mathcal{B}$ before and the label belongs to Chal. Such labels are called *new* labels. Let New be the list of pre-images to the new labels in the order as they appear in the queries. Let v be a bit string of length u that records the new queries of $\mathcal{B}$ such that the ith bit of v is 1 if and only if the ith query made by $\mathcal{B}$ is a *new* **add** query.

Given $\mathcal{B}$ we claim that any $\mathsf{Successful}$ labeling function **label** can be fully described by z, Chal_x, the index set $\{i_1, \ldots, i_w\}$, v, New and the outputs of **label** on all input points in $\mathbb{Z}_p \setminus \mathsf{Chal}_\mathsf{x}$. Indeed, for any $x \notin \mathsf{Chal}_\mathsf{x}$, the value $\mathbf{label}(x)$ is already given. We determine the labels of $x \in \mathsf{Chal}_\mathsf{x}$ as follows: run $\mathcal{B}^{\mathbf{label},\mathbf{add}}(g, z)$ to get Chal. We first explain how to reply to $\mathcal{B}$'s queries using the provided information.

1. Answering **label** queries of $\mathcal{B}$: By condition (ii), we know the answer does not appear in Chal, which means the input of the query does not appear in Chal_x. Since **label** is completely determined on $\mathbb{Z}_p \setminus \mathsf{Chal}_\mathsf{x}$, we can successfully answer such queries.
2. Answering **add** queries of $\mathcal{B}$: First note that by assumption, if the answer to the query is not $\perp$, then its pre-image must be in Chal_x, which means we can answer correctly assuming we know the pre-images to the input labels. In the following, we show how to find pre-images with the provided information. Using v, one can tell if the query is new.
   - Suppose the query is new. We then know both of the input labels are valid.
     - If one of the labels has pre-image in $\mathbb{Z}_p \setminus \mathsf{Chal}_\mathsf{x}$ or has been seen before, we can retrieve the pre-image of the other label in New.

- Otherwise, it must be the case that both labels are new and we can retrieve the pre-images in New.

– Suppose the query is not new.
- If the answer query to this query is not $\perp$, it must be the case that the labels either have pre-images in $\mathbb{Z}_p \setminus \mathsf{Chal}_\mathsf{x}$ or have been seen before, we can answer the queries directly.
- Otherwise, it must be the case that the answer to this query is $\perp$.

Thus, the set Chal can be retrieved. Once Chal is retrieved, sort its elements to get $(\ell_1, \dots \ell_t)$ and use the provided $(i_1, \dots, i_w)$ to retrieve $(\ell_{i_1}, \dots, \ell_{i_w})$. Assuming $\mathsf{Chal}_\mathsf{x} = (x_{i_1}, \dots, x_{i_w})$, we have $\mathbf{label}(x_{i_h}) = \ell_{i_h}$ for $h \in [w]$.

We now count f the number of bits required to describe Chal_x, the indices $\{i_1, \dots, i_w\}$ and $\mathbf{label}$'s outputs on all of $\mathbb{Z}_p \setminus \mathsf{Chal}_\mathsf{x}$. We can describe the sorted set Chal_x with $\log(\binom{p}{w} w!)$ bits. We can describe the index set with $\log \binom{t}{w}$ bits. We can describe the function $\mathbf{label}$ on $\mathbb{Z}_p \setminus \mathsf{Chal}_\mathsf{x}$ with $\log \frac{(s-w)!}{(s-p)!}$ bits. The string v has length u. The list New can be described with $\log w!$ bits because we can choose a permutation of the w pre-images whose initial items form the list New. □

B Attacks on RBE with CRS

B.1 TDP-Impossibility of PKCom with CRS

Theorem 8. *For $\epsilon := \frac{1}{\mathsf{poly}(\kappa)}$ let $\mathcal{E}^{\mathbf{O}} := (\mathsf{CRS}^{\mathbf{O}}, \mathsf{Key}^{\mathbf{O}}, \mathsf{Com}^{\mathbf{O}}, \mathsf{Enc}^{\mathbf{O}}, \mathsf{Dec}^{\mathbf{O}})$ be a $(1-\epsilon)$-correct PKCom scheme with respect to a random TDP oracle $\mathbf{O} = (\mathbf{g}, \mathbf{e}, \mathbf{d})$. Suppose a public parameter pp under $\mathcal{E}^{\mathbf{g},\mathbf{e},\mathbf{d}}$ satisfies $|\mathsf{pp}| \leq \frac{(n-2)|\mathsf{ik}|}{2}$, where n is the number of users and ik is a base index key (recall $|\mathsf{ik}| = 3\kappa$, Defintion 2). Also, let α be the number of queries made by $\mathsf{CRS}^{\mathbf{O}}(1^\kappa, 1^n)$ to the oracle $\mathbf{O}$. Then, there exists a $(1-\epsilon)(1-\frac{1}{\alpha})\frac{(1-2^{-\kappa/3})}{n}$-correct target-restricted signature scheme relative to $\mathbf{O} = (\mathbf{g}, \mathbf{e}, \mathbf{d})$*

We give the construction in Construction 9.

Construction 9 *We construct a n-target-restricted signature scheme from any PKCom scheme $\mathcal{E}^{\mathbf{O}} = (\mathsf{CRS}^{\mathbf{O}}, \mathsf{Key}^{\mathbf{O}}, \mathsf{Com}^{\mathbf{O}}, \mathsf{Enc}^{\mathbf{O}}, \mathsf{Dec}^{\mathbf{O}})$. The construction is parameterized over an integer s, which will be parameterized later; this parameter will only affect the size of the verification key. We assume all the algorithms satisfy the assumption in Note 1.*

– $\mathrm{Gen}^{\mathbf{O}}(1^\kappa, h) \to (\mathsf{sgk}, \mathsf{vrk})$ *where $h \in [n]$ is the message to be signed:*
1. *Run $\mathsf{CRS}^{\mathbf{O}}(1^\kappa, 1^n) \to \mathsf{crs}$ and let QCRS be the set of all Q-A pairs made to $\mathbf{g}$ and $\mathbf{e}$.*[15]
2. *For $1 \leq j \leq n$, run $\mathsf{Key}^{\mathbf{O}}(1^\kappa, \mathsf{crs}) \to (\mathsf{pk}_j, \mathsf{sk}_j)$. Let QGen_j be the set of all Q-A pairs made to $\mathbf{g}$ and $\mathbf{e}$.*

[15] We do not keep track of $\mathbf{d}$ queries because of Note 1.

Algorithm 1 SampleKeys(s)

Require: $h \in [n], \mathsf{crs}, \{\mathsf{pk}_i\}_{i \neq h}$
  $\mathsf{K} \leftarrow \phi$
  $j \leftarrow 0$
  while $j < s$ **do**
    $j \leftarrow j + 1$
    $(\mathsf{pk}_h, \mathsf{sk}_h) \leftarrow \mathsf{Key}(1^\kappa, \mathsf{crs})$
    $\mathsf{pp} \leftarrow \mathsf{Com}(\mathsf{crs}, \mathsf{pk}_1, \ldots, \mathsf{pk}_h, \ldots, \mathsf{pk}_n)$
    $(m, \mathsf{ct}) \leftarrow \mathsf{Enc}(\mathsf{pp}, h)$
    run $\mathsf{Dec}(\mathsf{crs}, \mathsf{sk}_h, \mathsf{ct})$:
    for $\mathsf{qu} = \mathbf{d}(\mathsf{tk}, y)$ **do**:
      $\mathsf{ik} \leftarrow \mathbf{g}(\mathsf{tk})$
      add $(\mathsf{tk}, \mathsf{ik})$ to K
    end for
  end while
  return K

3. *Run* $\mathsf{Com}^{\mathbf{O}}(\mathsf{crs}, \mathsf{pk}_1, \ldots, \mathsf{pk}_n) \to \mathsf{pp}$ *and let* QCMP *be the set of all query response pairs made to* $\mathbf{g}$ *and* $\mathbf{e}$.
4. *Run* $\mathsf{SampleKeys}(\mathsf{crs}, h, \{\mathsf{pk}_i\}_{i \neq h})$ *as defined in Algorithm 1 to obtain a set* K.
5. *Return* $\mathsf{vrk} = ((\mathsf{pk}_1, \ldots, \mathsf{pk}_n), \cup_{j \neq h} \mathsf{QGen}_j \cup \mathsf{QCMP}, \mathsf{K})$, $\mathsf{sgk} = (\mathsf{sk}_h, \mathsf{QGen}_h, \mathsf{QCRS})$.

– $\mathrm{Sig}(\mathsf{sgk}, h) \to \sigma$*: For* sgk *as above, return* $\sigma = (\mathsf{sk}_h, \mathsf{QGen}_h, \mathsf{QCRS})$.
– $\mathrm{Ver}^{\mathbf{g},\mathbf{e},\mathbf{d}}(\mathsf{vrk}, \sigma, h) = \mathrm{Ver1}(\mathrm{Ver0}^{O}(\mathsf{vrk}, h), \sigma)$*: Parse* $\mathsf{vrk} := ((\mathsf{pk}_1, \ldots, \mathsf{pk}_n), \mathsf{S}, \mathsf{K})$ *and* $\sigma := (\mathsf{sk}_h, \mathsf{QGen}_h, \mathsf{QCRS})$.
1. $\mathrm{Ver0}^{\mathbf{g},\mathbf{e},\mathbf{d}}(\mathsf{vrk}, h) \to \alpha := (\mathsf{vrk}, h, m, c, \mathsf{QEnc})$*, where* $(m, c) \leftarrow \mathsf{Enc}^{\mathbf{g},\mathbf{e},\mathbf{d}}(\mathsf{pp}, h)$ *and* QEnc *is the set of all Q-A pairs made to* $\mathbf{g}$ *and* $\mathbf{e}$.
2. $\mathrm{Ver1}(\alpha, \sigma)$*: Retrieve* QEnc, S *and* K *from* α*. (Recall* $\mathsf{S} = \cup_{j \neq h} \mathsf{QGen}_j \cup \mathsf{QCMP} \cup \mathsf{K}$ *is in* vk*.) Parse* $\sigma := (\mathsf{sk}_h, \mathsf{QGen}_h, \mathsf{QCRS})$*. Let* $\mathsf{All} = \mathsf{S} \cup \mathsf{QEnc} \cup \mathsf{QGen}_h \cup \mathsf{QCRS}$*. Run* $\mathsf{DecSim}(\mathsf{crs}, h, \mathsf{sk}_h, \{\mathsf{pk}_i\}, c, (\mathsf{All}, \mathsf{QEnc}, \mathsf{QGen}_h, \mathsf{QCRS}))$*, which simulates the execution of* $\mathsf{Dec}^{O}(\mathsf{crs}, h, \mathsf{sk}_h, \{\mathsf{pk}_i\}, c)$ *by rendering queries via* $(\mathsf{All}, \mathsf{QEnc}, \mathsf{QGen}_h, \mathsf{QCRS})$*, as follows:*
   (a) *For a given* $\mathbf{g}$ *or* $\mathbf{e}$ *query, if the answer is already provided in* All*, reply with that answer; else, with a random string* z *of appropriate length. In case of answering with a random response, add the Q-A pair to* Fake *(initially empty).*[16]
   (b) *For a given query* $\mathsf{qu} := ((\mathsf{tk}, y) \underset{\mathbf{d}}{\to} ?)$*, if for some* ik, $(\mathsf{tk} \underset{\mathbf{g}}{\to} \mathsf{ik}) \in (\mathsf{All} \cup \mathsf{Fake})/(\mathsf{QGen}_h \cup \mathsf{QCRS})$ *and* $((\mathsf{ik}, x) \underset{\mathbf{e}}{\to} c) \in \mathsf{All}$ *for some* x*, respond to the query with* x*. Else, if for some* ik, $(\mathsf{tk} \underset{\mathbf{g}}{\to} \mathsf{ik}) \in \mathsf{QGen}_h \cup \mathsf{QCRS}$ *and* $((\mathsf{ik}, x) \underset{\mathbf{e}}{\to} y) \in (\mathsf{All}/\mathsf{QEnc}) \cup \mathsf{Fake}$ *for some* x*, respond to the query with* x*. Else, respond to the query with a random value* $r \leftarrow \{0,1\}^\kappa$.
   Letting m' *be the output of* DecSim*, output 1 if* $m' = m$ *and 0 otherwise.*

[16] Duplicate queries will be replied to with the same random response.

Lemma 10. *Suppose*

$$(\mathsf{Pub}_h \cap \mathsf{PubC}) \subseteq \cup_{i \neq h} \mathsf{Pub}_i. \tag{7}$$

Then by setting $s = \alpha^2$, where α is the number of queries made by $\mathsf{CRS}(1^\kappa, 1^n)$*, the probability that the algorithm* Ver *(Construction 9) outputs the correct bit is at least* $(1 - 2^{-2\kappa/3})(1 - \frac{1}{\alpha})$.

Proof. The proof is similar to the proof of Proposition 2.

For α_i, β_j defined in Proposition 2, we can use the same arguments to prove that with probability at least $1 - \mathsf{poly}(\kappa).2^{-\kappa}$, α_i and β_j are distinct, and none of them occur in QEnc.

An inconsistency may occur if the response to some query is incorrect according to what is entailed by $\cup_j \mathsf{QGen}_j \cup \mathsf{QCMP} \cup \mathsf{QCRS} \cup \mathsf{QEnc}$.

All $\mathbf{g}$ and $\mathbf{e}$ type queries will be answered according to All. Thus, we will not run into any inconsistency.

For a query $\mathsf{qu} := ((\mathsf{tk}, y) \underset{\mathbf{d}}{\rightarrow} ?)$, we might run into an inconsistency if

1. for some ik, $(\mathsf{tk} \underset{\mathbf{g}}{\rightarrow} \mathsf{ik}) \in \mathsf{All}$, and
2. $((\mathsf{ik}, \widetilde{x}) \underset{\mathbf{e}}{\rightarrow} y) \in \mathsf{All}$, and
3. $\widetilde{x} \neq x$ where x is the generated response for qu by DecSim.

Call this event Bad. The event Bad indeed causes a conflict because $(\mathsf{tk} \underset{\mathbf{g}}{\rightarrow} \mathsf{ik}) \in \mathsf{All}$ and $((\mathsf{ik}, \widetilde{x}) \underset{\mathbf{e}}{\rightarrow} y) \in \mathsf{All}$ will dictate a response $\widetilde{x}$ to a decryption query (tk, y), but a random response $x \neq \widetilde{x}$ was given.

We must have $(\mathsf{tk} \underset{\mathbf{g}}{\rightarrow} \mathsf{ik}) \notin \mathsf{All}/(\mathsf{QGen}_h \cup \mathsf{QCRS})$, because otherwise a random response would not have been generated. Therefore, $(\mathsf{tk} \underset{\mathbf{g}}{\rightarrow} \mathsf{ik}) \in \mathsf{QGen}_h \cup \mathsf{QCRS}$. If $(\mathsf{tk} \underset{\mathbf{g}}{\rightarrow} \mathsf{ik}) \in \mathsf{QGen}_h$, then based on Equation 7 we will not run into any inconsistencies as proved in the Proposition 2.

Let Fail be the event that $(\mathsf{tk} \underset{\mathbf{g}}{\rightarrow} \mathsf{ik}) \in \mathsf{QCRS}$, $((\mathsf{ik}, \widetilde{x}) \underset{\mathbf{e}}{\rightarrow} y) \in \mathsf{QEnc}$, and $\widetilde{x} \neq x$. Note that Fail happens if $(\mathsf{tk} \underset{\mathbf{g}}{\rightarrow} \mathsf{ik}) \in \mathsf{QCRS}/\mathsf{K}$ because otherwise a random response would have not been generated.

For each key pair $(\mathsf{tk} \rightarrow \mathsf{ik}) \in \mathsf{QCRS}$, we define $s + 1$ variables as below: For $1 \leq i \leq s$, let X_i be a Bernoulli variable that equals 1 when $\mathbf{d}(\mathsf{tk}, *)$ appears in ith iteration of Algorithm 1 and 0 otherwise. Also, let $X_{s+1} = 1$ if $\mathbf{d}(\mathsf{tk}, *)$ appears in execution of $\mathsf{Dec}(\mathsf{crs}, h, \sigma_1, c)$ while running Ver_1 and 0 otherwise. $X_1, \ldots, X_{s+1}$ are independent and have the same probability of being 1. This is because all X_i, for $1 \leq i \leq s$, are output of the same randomized experiment and X_{s+1} can also be seen as a result of the same experiment. Let DQ be the set of all tk such that $\mathbf{d}(\mathsf{tk}, *)$ is queried in execution of $\mathsf{Dec}_{\mathsf{Sim}}(\mathsf{crs}, h, \sigma, \mathsf{vrk}, c)$ while running Ver_1.

Therefore, based on Lemma 4, we can conclude that for any key pair $(\mathsf{tk} \rightarrow \mathsf{ik}) \in \mathsf{QCRS}$,

$$\Pr[\mathsf{tk} \in \mathsf{DQ}/\mathsf{K}] = \Pr[X_1 = 0 \wedge \cdots \wedge X_s = 0 \wedge X_{s+1} = 1] \leq \frac{1}{s}$$

which is negligible for sufficiently large s. Therefore, the probability that for some $(\mathsf{tk} \underset{\mathbf{g}}{\rightarrow} \mathsf{ik}) \in \mathsf{QCRS}$, $(\mathsf{tk} \underset{\mathbf{g}}{\rightarrow} \mathsf{ik}) \notin \mathsf{K}$ but $\mathbf{d}(\mathsf{tk}, *)$ is queried while running Ver_1 is at most $\frac{\alpha}{s}$ where $\alpha = \mathsf{poly}(\kappa)$ is the number of queries made to the oracle by CRS to output crs. Let $s = \alpha^2$. Thus, Fail occurs with probability at most $\frac{1}{\alpha}$

The proof is now complete. □

Lemma 11. *Suppose Π is the signature scheme defined in Construction 9 with oracle access of the form $(\mathrm{Gen}^{\mathbf{g},\mathbf{e},\mathbf{d}}, \mathrm{Sig}, \mathrm{Ver}^{\mathbf{g},\mathbf{e},\mathbf{d}})$ and the PKCom scheme underlying Π is $(1-\epsilon)$-correct. Then, Π is $(1-\epsilon)(1-\frac{1}{\alpha})(\frac{1-2^{-\kappa/3}}{n})$-correct.*

Proof. Let $\mathsf{Success}$ be the event that the verification algorithm outputs the correct bit. In other words, if $\mathsf{Success}_i$ holds, we have: $\mathrm{Ver}^{\mathbf{g},\mathbf{e},\mathbf{d}}(\mathsf{vrk}, i, \mathrm{Sig}(\mathsf{sgk}, i)) = 1$ where $(\mathsf{sgk}, \mathsf{vrk}) \leftarrow \mathrm{Gen}^{\mathbf{g},\mathbf{e},\mathbf{d}}(1^\kappa, i)$. Finally, let X be the random variable denoting the message chosen to be signed. Also, let Good be the event that there exists $h \in [n]$ for which Proposition 1 holds. In other words, if Good happens, we have:

$$(\mathsf{Pub}_h \cap \mathsf{PubC}) \subseteq \cup_{j \neq h} \mathsf{Pub}_j$$

Based on Proposition 1, $\Pr[\mathsf{Good}] \geq 1 - 2^{-\kappa/2}$. Therefore:

$$\Pr[\Pi \text{ is correct}] = \sum_{i=1}^{n} \Pr[\mathsf{Success} | \mathsf{X} = i] \Pr[\mathsf{X} = i] = \frac{1}{n} \sum_{i=1}^{n} \Pr[\mathsf{Success}_i]$$

$$\Pr[\mathsf{Success}_i] = \Pr[\mathsf{Success}_i | \mathsf{Good}] \Pr[\mathsf{Good}] + \Pr[\mathsf{Success}_i | \overline{\mathsf{Good}}] \Pr[\overline{\mathsf{Good}}] \geq$$
$$\Pr[\mathsf{Success}_i | \mathsf{Good}] \Pr[\mathsf{Good}] \geq (1 - 2^{-\kappa/2}) \Pr[\mathsf{Success}_i | \mathsf{Good}]$$

Finally, let $\mathsf{Unlucky}$ be the event that there exists a $(\mathsf{tk} \underset{\mathbf{g}}{\rightarrow} \mathsf{ik}) \in \mathsf{QCRS} \wedge ((\mathsf{ik}, *) \underset{\mathbf{e}}{\rightarrow} *) \in \mathsf{QEnc}$ but $(\mathsf{tk} \underset{\mathbf{g}}{\rightarrow} \mathsf{ik}) \notin \mathsf{K}$. Based on Lemma 4, $\Pr[\mathsf{Unlucky}] \leq \frac{1}{\alpha}$.

$$\Pr[\Pi \text{ is correct}] = \frac{1}{n} \sum_{i=1}^{n} \Pr[\mathsf{Success}_i] \geq \frac{1}{n}(1 - 2^{-\kappa/2}) \sum_{i=1}^{n} \Pr[\mathsf{Success}_i | \mathsf{Good}] \geq$$

$$\frac{1}{n}(1-2^{-\kappa/2}) \Pr[\mathsf{Success}_h | \mathsf{Good}] \geq \frac{1}{n}(1-2^{-\kappa/2}) \Pr[\mathsf{Success}_h | \mathsf{Good} \wedge \overline{\mathsf{Unlucky}}] \Pr[\overline{\mathsf{Unlucky}}] \geq$$
$$\frac{1}{n}(1 - 2^{-\kappa/2})(1 - 2^{-2\kappa/3})(1 - \frac{1}{\alpha}) \geq (1 - \frac{1}{\alpha}) \frac{(1 - 2^{-\kappa/3})}{n}$$

□

Lemma 12. *Construction 9 is one-time unforgeable if the PKCom scheme is secure.*

Proof. Let $\mathcal{A}$ be a PPT adversary such that $\Pr[\mathrm{Ver}^{\mathbf{g},\mathbf{e},\mathbf{d}}(\mathsf{vrk}, h, \sigma) = 1$ where $\sigma \leftarrow \mathcal{A}^O(\mathsf{vrk}, h)]$ is non-negligible, where $h \leftarrow [n]$, and $(\mathsf{sgk}, \mathsf{vrk}) \leftarrow \mathrm{Gen}^{\mathbf{g},\mathbf{e},\mathbf{d}}(1^\kappa, h)$. We construct an adversary $\mathcal{A}'$ to win the security game in Definition 1 with non-negligible advantage performing the following steps:

1. $\mathcal{A}'$ runs $\mathcal{A}$ and receives $h \in [n]$ as the challenge message to be signed by $\mathcal{A}$.
2. $\mathcal{A}'$ receives crs from its challenger $\mathcal{C}$ and runs 1 to obtain K.
3. $\mathcal{A}'$ runs $(\mathsf{pk}_j, \mathsf{sk}_j) \leftarrow \mathsf{Key}(1^\kappa, \mathsf{crs})$ for $j \in [n] \backslash h$ and adds all Q-A pairs to QGen_j. Then, it sends $(h, \{\mathsf{pk}_j\}_{j \in [n] \backslash h})$ to its challenger $\mathcal{C}$.
4. $\mathcal{A}'$ receives pk_h and ct from the challenger, runs $\mathsf{Com}(\mathsf{crs}, \mathsf{pk}_1, \ldots, \mathsf{pk}_n)$ and add all query answer pairs to QCMP.
5. $\mathcal{A}'$ sends $(\{\mathsf{pk}_j\}_{j=1}^n, \cup_{j \neq h} \mathsf{QGen}_j \cup \mathsf{QCMP}, \mathsf{K})$ to $\mathcal{A}$.
6. $\mathcal{A}'$ receives σ from $\mathcal{A}$, runs $\mathsf{Dec}_{\mathsf{Attack}}^{\mathbf{g},\mathbf{e},\mathbf{d}}(h, \sigma, \{\mathsf{pk}_j\}_{j \in [n]}, \cup_{j \neq h} \mathsf{QGen}_j \cup \mathsf{QCMP}, \mathsf{K}, \mathsf{ct})$ to obtain m_1 and outputs m_1.
   The algorithm $\mathsf{Dec}_{\mathsf{Attack}}$ is similar to Dec but answers queries in a slightly different way. Let σ_1 be the first part of σ. $\mathsf{Dec}_{\mathsf{Attack}}$ takes as an input $(h, \sigma, \{\mathsf{pk}_j\}_{j \in [n]}, \cup_{j \neq h} \mathsf{QGen}_j \cup \mathsf{QCMP}, \mathsf{K}, \mathsf{ct})$ and works as follows:
   Run $\mathsf{Dec}(\mathsf{crs}, h, \sigma_1, \mathsf{ct})$: For any $\mathbf{g}$ or $\mathbf{e}$ type query, call the oracle $\mathbf{O}$ and return the same response. Let $\mathsf{U} = \cup_{j \neq h} \mathsf{QGen}_j \cup \mathsf{QCMP} \cup \mathsf{K} \cup \mathsf{QGen}'_h \cup \mathsf{QCRS}'$. For a given query $\mathbf{d}(\mathsf{tk}, y)$, if for no ik, $(\mathsf{tk} \underset{\mathbf{g}}{\rightarrow} \mathsf{ik}) \in \mathsf{U}$, then query $\mathbf{d}(\mathsf{tk}, y)$ to obtain x' and respond with x'. If for some ik, $(\mathsf{tk} \underset{\mathbf{g}}{\rightarrow} \mathsf{ik}) \in \mathsf{U}/(\mathsf{QGen}'_h \cup \mathsf{QCRS}')$ and for some x', $((\mathsf{ik}, x') \underset{\mathbf{e}}{\rightarrow} y) \in \mathsf{QGen}'_h \cup \mathsf{QCRS}'$, then respond with x'. Else, if for some ik, $(\mathsf{tk} \underset{\mathbf{g}}{\rightarrow} \mathsf{ik}) \in \mathsf{U}/(\mathsf{QGen}'_h \cup \mathsf{QCRS}')$, then query $\mathbf{d}(\mathsf{tk}, y)$ to obtain x' and respond with x'.
   Else, if for some ik, $(\mathsf{tk} \underset{\mathbf{g}}{\rightarrow} \mathsf{ik}) \in \mathsf{QGen}'_h \cup \mathsf{QCRS}'$ and for some x', $((\mathsf{ik}, x') \underset{\mathbf{e}}{\rightarrow} y) \in \mathsf{U}$, then respond with x'. Else, respond to the query with a random value $r \leftarrow \{0,1\}^\kappa$.

Let QEnc be the set of all Q-A pairs made during the execution of $\mathsf{Enc}^{\mathbf{g},\mathbf{e},\mathbf{d}}(\mathsf{pp}, h) \rightarrow (m_0, \mathsf{ct})$ by the challenger $\mathcal{C}$. Therefore, $\mathsf{All}' = \mathsf{U} \cup \mathsf{QEnc}$. Also, let QDec be the set of all Q-A pairs generated during the execution of $\mathsf{Dec}_{\mathsf{attack}}(\mathsf{crs}, h, \sigma, \{\mathsf{pk}_j\}_{j \in [n]}, \cup_{j \neq h} \mathsf{QGen}_j \cup \mathsf{QCMP}, \mathsf{K}, \mathsf{ct}) \rightarrow m_1$ and QDec' be the set of all Q-A pairs made during the execution of $\mathsf{Dec}_{\mathsf{Sim}}(\mathsf{crs}, h, \sigma, \{\mathsf{pk}_j\}_{j \in [n]}, \cup_{j \neq h} \mathsf{QGen}_j \cup \mathsf{QCMP}, \mathsf{QEnc}, \mathsf{ct}) \rightarrow m_2$ as defined in Construction 5.

Now, consider the two distributions
$\mathsf{D} : (\mathsf{QCRS}, \mathsf{QGen}_{j \neq h} \cup \mathsf{QCMP}, \{\mathsf{pk}_j\}_{j \in [n]}, \mathsf{QEnc}, \mathsf{ct}, \sigma, \mathsf{QDec}, m_1)$ and
$\widetilde{\mathsf{D}} : (\mathsf{QCRS}', \mathsf{K}, \mathsf{QGen}_{j \neq h} \cup \mathsf{QCMP}, \{\mathsf{pk}_j\}_{j \in [n]}, \mathsf{QEnc}, \mathsf{ct}, \sigma, \mathsf{QDec}', m_2)$. We claim that D and $\widetilde{\mathsf{D}}$ are almost identical meaning that for any event E, we have:

$$|\Pr[\mathsf{E} \text{ holds for } \mathsf{D}] - \Pr[\mathsf{E} \text{ holds for } \widetilde{\mathsf{D}}]| \leq \mathsf{neg}(\kappa)$$

where $\mathsf{neg}(\kappa)$ is a negligible function with respect to κ.

Let $\mathbf{d}_{\mathsf{Sim}}, \mathbf{d}_{\mathsf{attack}}$ denote the answers to a decryption query provided by $\mathsf{DecSim}, \mathsf{Dec}_{\mathsf{Attack}}$, respectively. For a given decryption query (tk, c), there are six possible cases:

– There exists a ik such that $(\mathsf{tk} \underset{\mathbf{g}}{\rightarrow} \mathsf{ik}) \in \mathsf{U}/(\mathsf{QGen}'_h \cup \mathsf{QCRS}')$ and there exists x such that $((\mathsf{ik}, x) \underset{\mathbf{e}}{\rightarrow} y) \in \mathsf{QGen}'_h \cup \mathsf{QCRS}'$: In this case the response to both $\mathbf{d}_{\mathsf{Sim}}(\mathsf{tk}, y)$,$\mathbf{d}_{\mathsf{attack}}(\mathsf{tk}, y)$ will be x.

– There exists a ik such that $(\mathsf{tk} \underset{\mathbf{g}}{\rightarrow} \mathsf{ik}) \in \mathsf{U}/(\mathsf{QGen}'_h \cup \mathsf{QCRS}')$ and there exists x such that $((\mathsf{ik}, x) \underset{\mathbf{e}}{\rightarrow} y) \in \mathsf{QEnc} \cup_{j \neq h} \mathsf{QGen}_j \cup \mathsf{QCMP}$: In this case the response to $\mathbf{d}_{\mathsf{Sim}}(\mathsf{tk}, y)$ will be x and the response to $\mathbf{d}_{\mathsf{attack}}(\mathsf{tk}, y)$ will be $\mathbf{d}(\mathsf{tk}, y)$ which equals x since the oracle $\mathbf{O}$ is correct.
– There exists a ik such that $(\mathsf{tk} \underset{\mathbf{g}}{\rightarrow} \mathsf{ik}) \in \mathsf{QGen}'_h \cup \mathsf{QCRS}'$ and for some x, we have $((\mathsf{ik}, x) \underset{\mathbf{e}}{\rightarrow} y) \in \mathsf{U}$: In this case the response to both $\mathbf{d}_{\mathsf{Sim}}(\mathsf{tk}, y)$, $\mathbf{d}_{\mathsf{attack}}(\mathsf{tk}, y)$ will be x.
– For some ik, $(\mathsf{tk} \underset{\mathbf{g}}{\rightarrow} \mathsf{ik}) \in \mathsf{QGen}'_h \cup \mathsf{QCRS}'$ and for no x, we have $((\mathsf{ik}, x) \underset{\mathbf{e}}{\rightarrow} y) \in \mathsf{U}$: In this case, the response to both $\mathbf{d}_{\mathsf{Sim}}(\mathsf{tk}, y)$, $\mathbf{d}_{\mathsf{attack}}(\mathsf{tk}, y)$ will be chosen at random. Therefore, both will have the same distribution.
– For some ik, $(\mathsf{tk} \underset{\mathbf{g}}{\rightarrow} \mathsf{ik}) \in \mathsf{QEnc}$ and for some x, we have $((\mathsf{ik}, x) \underset{\mathbf{e}}{\rightarrow} y) \in \mathsf{All}'$, then the response to $\mathbf{d}_{\mathsf{Sim}}(\mathsf{tk}, y)$ will be x and the response to $\mathbf{d}_{\mathsf{attack}}(\mathsf{tk}, y)$ will be $\mathbf{d}(\mathsf{tk}, y)$ which equals x since the oracle $\mathbf{O}$ is correct.
– For some ik, $(\mathsf{tk} \underset{\mathbf{g}}{\rightarrow} \mathsf{ik}) \in \mathsf{QEnc}$ but for no x, we have $((\mathsf{ik}, x) \underset{\mathbf{e}}{\rightarrow} y) \in \mathsf{All}'$, then the response to $\mathbf{d}_{\mathsf{Sim}}(\mathsf{tk}, y)$ will be chosen at random, while the response to $\mathbf{d}_{\mathsf{attack}}(\mathsf{tk}, y)$ will be $\mathbf{d}(\mathsf{tk}, y)$. Note that since for no x, $((\mathsf{ik}, x) \underset{\mathbf{e}}{\rightarrow} y) \in \mathsf{All}'$, the query doesn't make a conflict between the two distributions because there is no way to distinguish between $\mathbf{d}(\mathsf{tk}, y)$ and a random value for a distinguisher who only has access to All'.

Now let E be the event that m_1, the decryption of ct using the simulated decryption algorithm, equals m_0. Since D and $\widetilde{\mathsf{D}}$ are almost identical, if E holds for $\widetilde{\mathsf{D}}$, it will also hold for D with all but negligible probability. Considering that $\mathcal{A}$ has non-negligible advantage in forging a valid signature, i.e. one that is accepted with the algorithm Ver defined in Construction 9, m_2 will be equal to m_0, with non-negligible probability. Therefore, E holds for $\widetilde{\mathsf{D}}$ as well as D and $m_1 = m_0$ with non-negligible probability which contradicts security of the PKCom scheme. □

B.2 Impossibility of PKCom with CRS in Shoup's GGM

Now, we present the transformation of PKCom to target-restricted signatures while allowing CRS.

Theorem 10. *If there exists a $(1-\epsilon)$-correct PKCom scheme $(\mathsf{CRS}^{\mathbb{G}_{RR}}, \mathsf{Key}^{\mathbb{G}_{RR}}, \mathsf{Com}^{\mathbb{G}_{RR}}, \mathsf{Enc}^{\mathbb{G}_{RR}}, \mathsf{Dec}^{\mathbb{G}_{RR}})$ in the RR generic group model, then there exists a δ- correct target-restricted signature scheme in the same model where $\delta = (1-\epsilon)\frac{(1-2^{-\kappa/3})}{n}$.*

Construction 11 *We construct a target-restricted signature scheme defined over messages in $[n]$ from any PKCom scheme in the following way.*

– $\mathrm{Gen}^{\mathbb{G}_{RR}}(1^\kappa, h) \rightarrow (\mathsf{sgk}, \mathsf{vrk})$ *where $h \in [n]$ is the message to be signed. For $i \in [n]$ let $\mathsf{QGen}_i = \emptyset$.*

1. *Run* $\mathsf{CRS}^{\mathbb{G}_{RR}}(1^\kappa, 1^n) \to \mathsf{crs}$ *and all Q-A pairs made to* $\mathbb{G}_{RR}$ *to* QCRS.
2. *For* $1 \le j \le n$, *run* $\mathsf{Key}^{\mathbb{G}_{RR}}(1^\kappa, \mathsf{crs}) \to (\mathsf{pk}_j, \mathsf{sk}_j)$ *and add all Q-A pairs made to* $\mathbb{G}_{RR}$ *to* QGen_j.
3. *Run* $\mathsf{Com}^{\mathbb{G}_{RR}}(\mathsf{crs}, \mathsf{pk}_1, \ldots, \mathsf{pk}_n) \to \mathsf{pp}$ *and let* QCMP *be the set of all Q-A pairs made to* $\mathbb{G}_{RR}$.
4. *Update* $\mathsf{Known} \leftarrow \mathsf{Upd}(\cup_{i \neq h} \mathsf{QGen}_i \cup \mathsf{QCMP} \cup \mathsf{QCRS})$ *(Definition 11).*
5. *Return* $\mathsf{vrk} = ((\mathsf{pk}_1, \ldots, \mathsf{pk}_n), \cup_{j \neq h} \mathsf{QGen}_j \cup \mathsf{QCMP}, \mathsf{Known}, \upsilon)$, $\mathsf{sgk} = (\mathsf{sk}_h, \mathsf{QGen}_h, \mathsf{QCRS})$.

– $\mathrm{Sig}(\mathsf{sgk}, h) \to \sigma$*: For* sgk *as above, return* $\sigma := (\mathsf{sk}_h, \mathsf{QGen}_h, \mathsf{QCRS})$.

– $\mathrm{Ver}^{\mathbb{G}_{RR}}(\mathsf{vrk}, \sigma, h) = \mathrm{Ver1}(\mathrm{Ver0}^{\mathbb{G}_{RR}}(\mathsf{vrk}, h), \sigma)$: *Parse* $\mathsf{vrk} := ((\mathsf{pk}_1, \ldots, \mathsf{pk}_n), \mathsf{A}, \mathsf{Known}, \upsilon)$ *and* $\sigma := (\mathsf{sk}_h, \mathsf{QGen}_h, \mathsf{QCRS})$.

1. $\mathrm{Ver0}^{\mathbb{G}_{RR}}(\mathsf{vrk}, h) \to \alpha := (\mathsf{vrk}, h, m, c, \mathsf{QEnc})$, *where* $(m, c) \leftarrow \mathsf{Enc}^{\mathbb{G}_{RR}}(\mathsf{pp}, h)$ *and* QEnc *is the set of all Q-A pairs made to* $\mathbb{G}_{RR}$.
2. $\mathrm{Ver1}(\alpha, \sigma)$: *Retrieve* QEnc, A *and* Known *from* α. *Recall* $\mathsf{A} = \cup_{j \neq h} \mathsf{QGen}_j \cup \mathsf{QCMP}$. *Update* $\mathsf{Known} \leftarrow \mathsf{Upd}(\mathsf{QEnc})$. *Let* $\mathsf{All} = \cup_{j \neq h} \mathsf{QGen}_j \cup \mathsf{QCMP} \cup \mathsf{QEnc}$. *Run* DecSim *which simulates the execution of* $\mathsf{Dec}^{\mathbb{G}_{RR}}(\mathsf{crs}, h, \mathsf{sk}_h, \{\mathsf{pk}_i\}, c)$ *by rendering queries via* $(\mathsf{All}, \mathsf{QGen}_h, \mathsf{QCRS})$, *as follows: Initialize two sets* $\mathsf{E} = \mathsf{Eq}(\mathsf{All})$ *and* $\mathsf{V} = \mathsf{Var}(\mathsf{All})$. *For a given query* $\mathbf{add}(\ell_1, \ell_2)$ *do the following:*
   (a) *If* $\ell_1 \notin \mathsf{V} \cup \mathsf{Var}(\mathsf{QGen}_h \cup \mathsf{QCRS})$ *or* $\ell_2 \notin \mathsf{V} \cup \mathsf{Var}(\mathsf{QGen}_h \cup \mathsf{QCRS})$, *respond to the query with* $\perp$.
   (b) *Else if both* $\ell_1, \ell_2 \in \mathsf{V}$, *if there exists* $\ell \in \mathsf{V} \cup \mathsf{Var}(\mathsf{QGen}_h \cup \mathsf{QCRS})$ *such that* $x_{\ell_1} + x_{\ell_2} - x_\ell \in \mathsf{Span}(\mathsf{E} \cup \mathsf{Eq}(\mathsf{QGen}_h \cup \mathsf{QCRS}))$, *return* ℓ. *If no such an* ℓ *is found, respond with a random label* ℓ', *add* $x_{\ell_1} + x_{\ell_2} - x_{\ell'}$ *to* E *and add* ℓ' *to* V. *Also, set* $\mathsf{Known}(\ell') = \top$.
   (c) *Else if there exists a label* ℓ *such that* $x_{\ell_1} + x_{\ell_2} - x_{\ell'} \in \mathsf{Span}(\mathsf{Eq}(\mathsf{QCMP} \cup_i \mathsf{QGen}_i \cup \mathsf{QCRS}))$, *return* ℓ;
   (d) *Else, if there exists a label* ℓ *such that* $\mathsf{Known}(\ell) = \top$ *and* $x_{\ell_1} + x_{\ell_2} - x_\ell \in \mathsf{Span}(\mathsf{E} \cup \mathsf{Eq}(\mathsf{QGen}_h \cup \mathsf{QCRS}))$, *return* ℓ. *Else, respond with a random label* ℓ' *and add* $x_{\ell_1} + x_{\ell_2} - x_{\ell'}$ *to* E, *and add* ℓ' *to* V. *Also, set* $\mathsf{Known}(\ell') = \top$.

Lemma 13. *Suppose we update* $\mathsf{Known} \leftarrow \mathsf{Upd}(\mathsf{QCRS} \cup_{i \neq h} \mathsf{QGen}_i \cup \mathsf{QCMP} \cup \mathsf{QEnc})$. *Then, assuming*

$$((\mathsf{V}_h \cup \mathsf{VCMP}_h) \cap \mathsf{VQ}) \subseteq \cup_{i \neq h} (\mathsf{V}_i \cup \mathsf{VCMP}_i), \tag{8}$$

with probability at least $1 - \mathsf{poly}(\kappa).2^{-2\kappa}$ *for any label that* $\ell \in \mathsf{Var}(\mathsf{QCRS} \cup_{i \neq h} \mathsf{QGen}_i \cup \mathsf{QCMP} \cup \mathsf{QEnc})$ *but* $\ell \notin \mathsf{VCMP}_h$, $\mathsf{Known}(\ell) = \top$.

Proof. Suppose there exists a label $\ell \in \mathsf{Var}(\mathsf{QCRS} \cup_{i \neq h} \mathsf{QGen}_i \cup \mathsf{QCMP} \cup \mathsf{QEnc})$ such that $\ell \notin \mathsf{VCMP}_h$ and $\mathsf{Known}(\ell) = \perp$. If $\ell \in \mathsf{Var}(\mathsf{QCRS})$, then as per Definition 9, there exists a Q-A pair $((\ell_x, \ell_y) \xrightarrow[\mathbf{add}]{} \ell) \in \mathsf{QCRS}$. Let ℓ be the first label in $\mathsf{Var}(\mathsf{QCRS})$ such that $\mathsf{Known}(\ell) = \perp$. We must have $\mathsf{Known}(\ell_x) = \perp$ and $\mathsf{Known}(\ell_y) = \perp$ because otherwise $\mathsf{Known}(\ell) = \top$. Since ℓ is the first label in $\mathsf{Var}(\mathsf{QCRS})$ whose $\mathsf{Known} = \perp$, we must have $\ell_x \notin \mathsf{Var}(\mathsf{QCRS}) \wedge \ell_y \notin \mathsf{Var}(\mathsf{QCRS})$.

Moreover, ℓ_x, ℓ_y must be valid because otherwise, $\ell = \perp$ and $\ell \notin \mathsf{Var}(\mathsf{QCRS})$. Based on Proposition 4, the probability that $\ell_x \notin \mathsf{Var}(\mathsf{QCRS}) \wedge \ell_y \notin \mathsf{Var}(\mathsf{QCRS})$ but ℓ_x, ℓ_y are valid is at most $2^{-4\kappa}$.
If $\ell \notin \mathsf{Var}(\mathsf{QCRS})$, considering that for all labels $\ell' \in \mathsf{Var}(\mathsf{QCRS})$, $\mathsf{Known}(\ell') = \top$, Lemma 7 proves that with probability at least p_1, $\mathsf{Known}(\ell) = \top$.
Therefore, with probability at least $p_2 = 1 - \mathsf{poly}(\kappa).2^{-2^{\kappa}}$ for any label that $\ell \in \mathsf{Var}(\mathsf{QCRS} \cup_{i \neq h} \mathsf{QGen}_i \cup \mathsf{QCMP} \cup \mathsf{QEnc})$ but $\ell \notin \mathsf{VCMP}_h$, $\mathsf{Known}(\ell) = \top$. □

Lemma 14. *Assuming*

$$((\mathsf{V}_h \cup \mathsf{VCMP}_h) \cap \mathsf{VQ}) \subseteq \cup_{i \neq h}(\mathsf{V}_i \cup \mathsf{VCMP}_i), \tag{9}$$

as defined in Definition 12, the probability that the algorithm Ver *(Construction 11) doesn't output the correct bit is at most* $\frac{(1-2^{-\kappa/3})}{n}$.

Proof. The proof is similar to the proof of Proposition 5. If we run to no inconsistencies, the decryption will be done correctly. Suppose $\mathsf{qu} = \mathbf{add}(\ell_1, \ell_2)$ is the first query inside DecSim whose response creates an inconsistency as per Definition 13. Let ℓ' be the response generated by DecSim and view $(\cup \mathsf{QGen}_i \cup \mathsf{QCMP} \cup \mathsf{QEnc} \cup \mathsf{QCRS} \cup \mathsf{T})$ where T is the previous set of Q-A pairs in DecSim as Se in Definition 13. It can be proved that qu cannot result in $\mathsf{Inconsistent}_1$ using the same argument as Proposition 5.

If qu results in $\mathsf{Inconsistent}_2$, then:

1. $x_{\ell_1} + x_{\ell_2} - x_{\ell^*}$ must be in $\mathsf{Span}(\mathsf{Eq}(\cup \mathsf{QGen}_i \cup \mathsf{QCMP} \cup \mathsf{QEnc} \cup \mathsf{QCRS} \cup \mathsf{T}))$.
2. $\ell' \neq \ell^*$.

First we argue that in order for $\mathsf{Inconsistent}_2$ to occur, we must have $\ell^* \in \mathsf{Var}(\cup \mathsf{QGen}_\mathsf{i} \cup \mathsf{QCMP} \cup \mathsf{QEnc} \cup \mathsf{QCRS} \cup \mathsf{T})$. Let's call this condition Cond. Suppose not. Then, there is no Q-A pair such that $((\ell'_1, \ell'_2) \xrightarrow[\mathbf{add}]{} \ell^*) \in \cup \mathsf{QGen}_\mathsf{i} \cup \mathsf{QCMP} \cup \mathsf{QEnc} \cup \mathsf{QCRS} \cup \mathsf{T}$. Therefore, there must be a query $((\ell^*, x) \xrightarrow[\mathbf{add}]{} y) \in \mathsf{QCRS} \cup_{\mathsf{i}=1}^{\mathsf{n}} \mathsf{QGen}_\mathsf{i} \cup \mathsf{QCMP} \cup \mathsf{QEnc} \cup \mathsf{T}$ or $((x, \ell^*) \xrightarrow[\mathbf{add}]{} y) \in \mathsf{QCRS} \cup_{\mathsf{i}=1}^{\mathsf{n}} \mathsf{QGen}_\mathsf{i} \cup \mathsf{QCMP} \cup \mathsf{QEnc} \cup \mathsf{T}$ because otherwise $x_{\ell_1} + x_{\ell_2} - x_{\ell^*}$ could have not been in $\mathsf{Span}(\mathsf{Eq}(\cup \mathsf{QGen}_i \cup \mathsf{QCMP} \cup \mathsf{QEnc} \cup \mathsf{QCRS} \cup \mathsf{T}))$. Since $\ell^* \notin \mathsf{Var}(\cup \mathsf{QGen}_\mathsf{i} \cup \mathsf{QCMP} \cup \mathsf{QEnc} \cup \mathsf{QCRS} \cup \mathsf{T})$, Proposition 4 implies that ℓ^* is invalid with probability $1 - 2^{-2\kappa}$. Therefore, $y = \perp$ with probability $1 - 2^{-2\kappa}$. Since equations containing $\perp$ will not be added to the $\mathsf{Eq}(*)$, no equation containing x_{ℓ^*} will be added to $\mathsf{Eq}(\cup \mathsf{QGen}_i \cup \mathsf{QCMP} \cup \mathsf{QEnc} \cup \mathsf{QCRS} \cup \mathsf{T})$ which contradicts $x_{\ell_1} + x_{\ell_2} - x_{\ell^*} \in \mathsf{Span}(\mathsf{Eq}(\cup \mathsf{QGen}_i \cup \mathsf{QCMP} \cup \mathsf{QEnc} \cup \mathsf{QCRS} \cup \mathsf{T}))$. Therefore, if Cond does not hold, the probability that qu causes $\mathsf{Inconsistent}_2$ is at most $2^{-2\kappa}$.

Now, suppose there exist a Q-A pair $((\ell'_1, \ell'_2) \xrightarrow[\mathbf{add}]{} \ell^*) \in \mathsf{QCRS} \cup \mathsf{QGen}_i \cup \mathsf{QCMP} \cup \mathsf{QEnc} \cup \mathsf{T}$.

We consider all possible cases:

1. $\ell^* \in \mathsf{Var}(\mathsf{QEnc})$: Suppose $((\ell'_1, \ell'_2) \xrightarrow[\mathbf{add}]{} \ell^*) \in \mathsf{QEnc}$. If both $\ell_1, \ell_2 \in \mathsf{V}$, then considering Condition 1, by Item 2b of DecSim, $\ell' = \ell^*$. Else, we claim that

$\mathsf{Known}(\ell^*) = \top$; thus, considering Condition 1, by Item 2d of DecSim, $\ell' = \ell^*$. If $\mathsf{Known}(\ell^*) = \bot$, then $\mathsf{Known}(\ell'_1) = \bot$ and $\mathsf{Known}(\ell'_2) = \bot$. Recall updating Known in 2 and 4. Based on Lemma 13, if $\ell'_1 \in \mathsf{Var}(\cup_{i\neq h}\mathsf{QGen}_i \cup \mathsf{QCRS} \cup \mathsf{QCMP} \cup \mathsf{QEnc})\backslash\mathsf{VCMP}_h$, with probability p_2, we must have $\mathsf{Known}(\ell'_1) = \top$. Suppose $\ell'_1 \notin \mathsf{Var}(\mathsf{QCRS} \cup_{i\neq h} \mathsf{QGen}_i \cup \mathsf{QCMP} \cup \mathsf{QEnc})\backslash\mathsf{VCMP}_h$, thus there are two possible cases:

(a) $\ell'_1 \in \mathsf{Var}(\mathsf{QGen}_h)\cup\mathsf{VCMP}_h$: Considering Definition 12, $\ell'_1 \in (\mathsf{V}_h\cup\mathsf{VCMP}_h)\cap \mathsf{VQ}$ but $\ell'_1 \notin \cup_{i\neq h}(\mathsf{V}_i \cup \mathsf{VCMP}_i)$ which contradicts Equation 9.

(b) $\ell'_1 \notin \mathsf{Var}(\mathsf{All}\cup\mathsf{QGen}_h\cup\mathsf{QCRS})$: ℓ'_1 must be valid because otherwise $\ell^* = \bot$ and qu could not have caused $\mathsf{Inconsistent}_2$. Thus, by Proposition 4, the probability of this case happening is at most $2^{-2\kappa}$.

Therefore, the probability that $\mathsf{Known}(\ell'_1) = \bot$ and $\mathsf{Known}(\ell'_2) = \bot$ is at most $(1 - p_2 + 2^{-2\kappa})^2 = \mathsf{poly}(\kappa).2^{-4\kappa}$. Thus, the probability that $\ell^* \in \mathsf{Var}(\mathsf{QEnc})$ and $\mathsf{Known}(\ell^*) = \bot$ is bounded by $2^{-3\kappa}$.

2. $\ell^* \in \mathsf{Var}(\cup_i\mathsf{QGen}_i \cup \mathsf{QCMP} \cup \mathsf{QCRS})$: Suppose $((\ell'_1, \ell'_2) \xrightarrow[\mathbf{add}]{} \ell^*) \in \cup_i\mathsf{QGen}_i \cup \mathsf{QCMP} \cup \mathsf{QCRS}$. If $\ell_1 \in \mathsf{V} \wedge \ell_2 \in \mathsf{V}$, considering Condition 1 above, Item 2b of DecSim, implies that $\ell' = \ell^*$. Else, if $\ell_1 \in \mathsf{Var}(\mathsf{QGen}_h \cup \mathsf{QCRS})$ or $\ell_2 \in \mathsf{Var}(\mathsf{QGen}_h \cup \mathsf{QCRS})$, considering Condition 1 above, Item 2c of DecSim, implies that $\ell' = \ell^*$.
3. $\ell^* \in \mathsf{Var}(\mathsf{T})$: Since $\ell^* \notin \mathsf{Var}(\mathsf{All} \cup \mathsf{QGen}_\mathsf{h} \cup \mathsf{QCRS})$, ℓ^* must be one of the labels that are randomly generated by DecSim. Therefore, based on the construction of DecSim, $\mathsf{Known}(\ell^*) = \top$. Thus, if $\ell_1 \in \mathsf{V} \wedge \ell_2 \in \mathsf{V}$, considering Condition 1 above, Item 2b of DecSim, implies that $\ell' = \ell^*$ and if $\ell_1 \in \mathsf{Var}(\mathsf{QGen}_h\cup\mathsf{QCRS})$ or $\ell_2 \in \mathsf{Var}(\mathsf{QGen}_h\cup\mathsf{QCRS})$, considering Condition 1 above, Item 2d of DecSim, implies that $\ell' = \ell^*$.

Thus, if Cond happens, the probability that qu causes $\mathsf{Inconsistent}_2$, is at most $2^{-3\kappa}$. Therefore, Ver doesn't output the correct bit with probability at most $\mathsf{poly}(\kappa).(2^{-3\kappa} + 2^{-2\kappa}) \leq 2^{-2\kappa/3}$.

□

We will now obtain the following lemma.

Lemma 15. *Suppose Π is the signature scheme defined in Construction 11 with oracle access of the form $(\mathrm{Gen}^{\mathbb{G}_{RR}}, \mathrm{Sig}, \mathrm{Ver}^{\mathbb{G}_{RR}})$ and the PKCom scheme underlying Π is $(1-\epsilon)$-correct. Then, Π is $(1-\epsilon)\frac{(1-2^{-\kappa/3})}{n}$-correct.*