Indifferentiability for Public Key Cryptosystems

Mark Zhandry[1] and Cong Zhang[2]

[1] Princeton University, mzhandry@princeton.edu
[2] Rutgers University, cz200@cs.rutgers.edu

Abstract. We initiate the study of indifferentiability for public key encryption and other public key primitives. Our main results are definitions and constructions of public key cryptosystems that are indifferentiable from ideal cryptosystems, in the random oracle model. Cryptosystems include:
- Public key encryption;
- Digital signatures;
- Non-interactive key agreement.

Our schemes are based on standard public key assumptions. By being indifferentiable from an ideal object, our schemes satisfy any security property that can be represented as a single-stage game and can be composed to operate in higher-level protocols.

Keywords. Indifferentiability, Composition, Public key encryption, Random oracle model, Ideal cipher model.

1 Introduction

When designing a cryptographic system, it is difficult to predict how it will be used in practice and what security properties will be required of it. For example, if the larger system produces certain error messages, this can lead to chosen ciphertext attacks [6]. Perhaps a message is encrypted using random coins which themselves are derived from the message, as is used for de-duplication [31]. Maybe the secret key itself will be encrypted by the system, as is sometimes used in disk encryption. Or perhaps there was bad randomness generation on the hardware device, leading to secret keys or encryption randomness that is low-entropy or correlated across many instances.

Cryptographers have devised different security models to capture each of the scenarios above and more, each requiring different constructions to satisfy. However, seldom are these different security models considered in tandem, meaning that each application scenario may require a different scheme. Even worse, there are many potential security models that have yet to be considered; after all, the ability of software developers to use and abuse cryptosystems is endless, and it is difficult to predict what the next abuse will be.

With the above in mind, our goal is to develop a *single* construction for a given cryptographic concept that simultaneously captures any reasonable security property and can be composed to work in any reasonable larger protocol. As such, only a single instance of the scheme needs to be developed and then deployed in a variety of use cases, even those that have not been discovered yet.

Ideal Public Key Cryptosystems. Our starting point will be the random oracle model (ROM) [4], a common heuristic used in cryptography. Here, a hash function is assumed to be so well designed that the only reasonable attacks simply evaluate the hash function as a black box and gain nothing by trying to exploit the particular design. To capture this, the hash function is modeled as a truly random function, accessible by making queries to the function.

A random oracle truly is the "ideal" hash function: it is trivially one-way and collision resistant, the stand security notions for hash functions. But it is also much stronger: it is correlation intractable [9], a good extractor even for computational sources, and much more. When used in a larger system, random oracles can yield provably secure schemes even when standard security properties for hash functions are insufficient. In fact, the most efficient schemes in practice are only known to be secure using random oracles. As such, the ROM is ubiquitous in cryptography.

Other idealized models have been studied before. Examples include the the ideal cipher model [28], the generic group model [29], and more recently ideal *symmetric* key encryption [2]. However, no prior work considers idealized models for public key cryptosystems.

In this work, we define and construct the first ideal *public key* cryptosystems such as public key encryption and digital signatures. By being ideal, our schemes will immediately satisfy a wide class of security properties, including most studied in the literature. Our schemes will be proved to be ideal in the random oracle model using Maurer's indifferentiability framework [23], under general computational assumptions. We also show that certain classic relations among cryptographic objects also hold in the ideal setting, while discussing cases where such relations fail.

Our goal comes with interesting challenges: on one hand, public key schemes tend to require number-theoretic structure in order to attain the necessary functionality. On the other hand, ideal schemes by definition have essentially no structure. Therefore, our results require novel techniques, including bringing indifferentiability into the public key setting.

1.1 What Is An Ideal Public Key Scheme?

Now, we turn to our results. Our first result is to define, precisely, what an "ideal" public key cryptosystem is. For simplicity, in the following discussion, we will consider the case of two-party non-interactive key exchange (NIKE). Such a scheme consists of two algorithms. KEYGEN is a key generation algorithm run by each of two users. We will adopt the convention that the input to KEYGEN is the user's secret key SK, and the output is the corresponding public key PK. The two users then exchange their public keys. They then run $\mathsf{SHAREDKEY}$ to extract a common shared key. $\mathsf{SHAREDKEY}$ will take as input the public key for one user and the secret key for the other, and output a shared key K. The correctness requirement is that both users arrive at the same key:

$$\mathsf{SHAREDKEY}(\mathsf{PK}_1, \mathsf{SK}_2) = \mathsf{SHAREDKEY}(\mathsf{PK}_2, \mathsf{SK}_1)$$

whenever $\mathsf{PK}_1 = \mathsf{KEYGEN}(\mathsf{SK}_1)$ and $\mathsf{PK}_2 = \mathsf{KEYGEN}(\mathsf{SK}_2)$.

Any NIKE scheme will have the syntax above and the same correctness requirement. On the other hand, any given NIKE scheme may have additional structural properties that make it insecure in certain settings. For example, if multiple shared keys are generated using the same public key PK for a given user, the resulting shared keys may be correlated in some algebraic way. In order to be secure in the widest variety of settings, an ideal NIKE scheme should therefore not have any such additional structure over the minimum needed to ensure correctness.

In the case of existing idealized models, the idealization is simply a uniformly random choice of procedures subject to the mandatory correctness requirements. For example, a hash function has no correctness requirement except for determinism; as such its idealization is a random oracle. Likewise, a block cipher must be a (keyed) permutation, and the decryption functionality must be its inverse. As such, the ideal cipher is a random keyed permutation and its inverse.

Therefore, the natural way to model an ideal NIKE scheme is to have all algorithms be random functions. Of course, the correctness requirement means that there will be correlations between the algorithms. We take an ideal NIKE scheme to be two oracles KEYGEN, $\mathsf{SHAREDKEY}$ such that:

- $\mathsf{KEYGEN}(\mathsf{SK})$ is a random injection
- $\mathsf{SHAREDKEY}(\mathsf{PK}, \mathsf{SK})$ is a random injection, except that $\mathsf{SHAREDKEY}(\mathsf{PK}_1, \mathsf{SK}_2) = \mathsf{SHAREDKEY}(\mathsf{PK}_2, \mathsf{SK}_1)$ whenever $\mathsf{PK}_1 = \mathsf{KEYGEN}(\mathsf{SK}_1)$ and $\mathsf{PK}_2 = \mathsf{KEYGEN}(\mathsf{SK}_2)$ [3].

We emphasize that all functions are public and visible to the attacker.

1.2 Indifferentiability

Of course, just like a random oracle/generic group/ideal cipher, ideal NIKE cannot exist in the real world. This then begs the question: how do we design and rigorously argue that a NIKE scheme is so well designed that it can be treated as an ideal NIKE scheme in applications?

Barbosa and Farshim [2] offer one possible answer. They build a symmetric key encryption scheme from a hash function. Then, they show, roughly, that if the hash function is ideal (that is, a random oracle), then so is their encryption scheme. Our goal in this work will be to do the same for public key schemes: to build an ideal NIKE scheme assuming that a hash function H is a random oracle.

As in [2], formal justification of ideal security requires some care. Suppose we have a construction of an ideal NIKE scheme $(\mathsf{KEYGEN}, \mathsf{SHAREDKEY})$ in the random oracle model, meaning each of the algorithms makes queries to a random function H. In the case where H is *hidden* to the adversary, such a construction is almost trivial, and would essentially reduce to building symmetric key encryption from a PRF. However, Maurer [23] observed that this is *not* enough, since H is a *public* function and the adversary can query H as well.

Clearly, any construction $(\mathsf{KEYGEN}, \mathsf{SHAREDKEY})$ will now be distinguishable from the idealized algorithms, since the adversary can evaluate the algorithms for himself by making queries to H, and checking if the results are consistent with the oracles provided. Instead, what is needed is Maurer's notion of *indifferentiability*, which says that when $(\mathsf{KEYGEN}, \mathsf{SHAREDKEY})$ are ideal, it is possible to *simulate* H by a simulator S which can make queries to $(\mathsf{KEYGEN}, \mathsf{SHAREDKEY})$. In the real world, $(\mathsf{KEYGEN}, \mathsf{SHAREDKEY})$ are constructed from H per the specification. In the ideal world, $(\mathsf{KEYGEN}, \mathsf{SHAREDKEY})$ are the idealized objects, and H is simulated by making queries to the ideal objects. Indifferentiability requires that the two worlds are indistinguishable to an adversary that gets access to all the oracles.

Maurer shows that indifferentiability has many desirable properties: it composes well and will be *as good as* the ideal object in many settings (see Section 1.3 below for some limitations).

Therefore, our goal will be to build NIKE which is indifferentiable from ideal NIKE in the random oracle model. As indifferentiability has mostly been used in the symmetric key setting, this will require new techniques to bring indifferentiability into the public key world. Indeed, most works on indifferentiability build ideal objects with minimal correctness requirements: none in the case of random oracles, and bijectivity/injectivity in the case of ideal ciphers/symmetric key encryption. The case of public key cryptosystems requires significantly more structure for correctness. In fact, we face an immediate theoretical barrier:

[3] By the injectivity of KEYGEN, this is still a well-defined function.

Impagliazzo and Rudich[21] demonstrate that a random oracle is incapable of constructing something as structured as public key encryption, even ignoring the strong indifferentiability requirement.

Instead, we will obtain our needed structure using public key tools. However, public key tools come with *too* much structure: every term has an algebraic meaning which is not present in the idealized setting. Therefore, our goal will actually be to employ a novel combination of public key techniques *together with* random oracles in order to eliminate this extra structure. The result will be an indifferentiable NIKE scheme.

1.3 Discussion

Limitations. Before giving our constructions in detail, we briefly discuss limitations. Most importantly, idealized cryptosystems do not exist in the real world. Even more, Canetti, Goldreich, and Halevi [9] demonstrate that no concrete instantiation in the standard model is "as good as" an ideal object. Therefore, idealizations of cryptographic primitives are only heuristic evidence for security.

Nevertheless, the counter-examples are usually somewhat contrived, and do not apply in typical real-world settings. Indeed, in the case of hash functions, significant resources have been invested in analyzing their security, and the best attacks typically treat the hash function as a random oracle. As such, the random oracle appears to be a reasonable approximation to the real world in most settings. By building schemes from such strong hash functions and proving security using indifferentiability, such schemes are essentially "as good as" the ideal schemes, assuming the underlying hash function is ideal.

In fact, the random oracle model is widely used for the construction of new cryptosystems, as it allows security to be justified where no obvious concrete security property for hash functions would suffice. In these cases, the system is typically proven to satisfy the single security property considered. In our case, we are able to rely on the same heuristic treatment of hash functions, and attain ideal security.

Now, Ristenpart, Shacham, and Shrimpton [25] demonstrate the limitations of the indifferentiability framework. In particular, they show that indifferentiability is *insufficient* for proving security properties defined by *multi-stage* games. While this potentially precludes certain applications, indifferentiability is still sufficient to prove many security properties such as CCA-security, key-dependent-message and circular security in restricted settings (see [2] for discussion), bounded leakage resilience, and more. We also note that if all but one of the stages are independent of the ideal primitives, then indifferentiability is sufficient. This captures, for example, the usual modeling of deterministic public key encryption in the random oracle model [3]. Even more, in the case where multiple stages depend on the ideal primitives, Mittelbach [24] shows that indifferentiability is sufficient in some settings.

We leave as an interesting direction for future work building ideal public key schemes that can be proven secure in stronger models of indifferentiability such as reset indifferentiability [25] or context-restricted indifferentiability [22].

In light of the above, we need to interpret our results, which always rely on a random oracle. There are a couple of possibilities:

- The random oracle is instantiated with a concrete hash function. In this case, we know that the obtained scheme is not truly ideal in the real world. However, just as well-designed hash functions heuristically behave as random oracles, we can interpret our indifferentiability proof as strong heuristic evidence for security under a wide range of attack scenarios. Any new attack on our scheme would yield an interesting attack on the underlying hash function.

– The hash function is implemented by a pseudorandom function (PRF) whose key is controlled by a trusted third party. The third party offers an oracle interface to the PRF; by PRF security this is computationally indistinguishable from a random oracle. Our results show that such a simple trusted resource can be used to build more complicated ideal resources such as public key encryption and digital signatures.

Relationship to Universal Composability. There are some parallels between our idealized cryptosystems and *universal composability* (UC) [8]. Both seek to define an "ideal" object for a given cryptographic concept. Both consider composition, using one ideal functionality to build another. Both use a simulation to define security and composition.

However, the two notions are also fundamentally different. In UC, an ideal functionality is specified by considering how a trusted third party would solve a given cryptographic task. Then the actual protocol "emulates" this ideal through an interactive protocol. For example, the cryptographic task solved by encryption is simply private communication, and as such the ideal functionality is more or less message passing.

In contrast, in our setting, rather than consider the abstract task such as message passing, we consider the cryptographic abstraction of solving such a task concretely. So we consider, for example, public key encryption rather than the task of private communication. The ideal public key encryption scheme is then a random choice of functions, subject to the constraints imposed by the correctness requirements. The main benefit of our approach is that we naturally handle a much wider set of use cases such as key-dependent message security and weak/non-existent randomness; there is no natural way to model these use cases in the UC framework.

1.4 Constructing Ideal NIKE

We now turn to our constructions. Our goal will be to combine a *standard model* NIKE $(\mathsf{keygen}, \mathsf{sharedkey})$ — one with concrete mild security properties that are easy to instantiate — with random oracles to obtain an ideal model NIKE $(\mathsf{KEYGEN}, \mathsf{SHAREDKEY})$.

Making KEYGEN *indifferentiable.* First, we will focus just on KEYGEN, which on its own must be indifferentiable from a random injection. Of course, we could just set KEYGEN to be a random oracle[4], but we want to somehow incorporate keygen so that it can provide the structure needed when we turn to construct $\mathsf{ENC}, \mathsf{DEC}$.

Nevertheless, a random oracle (or some other idealized object) is needed somewhere in the construction. As a first attempt, we could consider defining $\mathsf{KEYGEN}(\mathsf{SK}) = H(\mathsf{keygen}(\mathsf{SK}))$, hashing the output of keygen to eliminate any structure on the public keys. This, unfortunately, does not work. For example, keygen may not be collision resistant, and any collision for keygen will therefore give a collision for KEYGEN. The resulting KEYGEN would then clearly be distinguishable from a random function without even making queries to H. Even if we assume keygen was collision resistant, the scheme would still not be indifferentiable. Indeed, the attacker could can query $\mathsf{KEYGEN}(\mathsf{SK})$, evaluate $\mathsf{pk} = \mathsf{keygen}(\mathsf{SK})$ for itself, and then query H on pk. The simulator now has to simulate H, and for indifferentiability to hold it must know how to set $H(\mathsf{pk}) = \mathsf{KEYGEN}(\mathsf{SK})$. However, the simulator only gets to see

[4] By having the random oracle be sufficiently expanding, it will be an injection with high probability.

pk and somehow must query KEYGEN on SK. Extracting SK from pk involves breaking the original NIKE, which is presumably intractable.

A different approach would be to define $\mathsf{KEYGEN}(\mathsf{SK}) = \mathsf{keygen}(H(\mathsf{SK}))$. The problem here is that keygen may output very structured public keys, which are clearly distinguishable from random. We could assume that keygen has pseudorandom public keys; that is, that keygen applied to uniformly random coins gives a pseudorandom output. However, we still have a problem. Indeed, suppose the adversary queries $\mathsf{KEYGEN}(\mathsf{SK})$, which in the ideal world will give a random string. Then the adversary queries $H(\mathsf{SK})$. In the ideal world, the simulator must set $H(\mathsf{SK}) = r$ such that $\mathsf{keygen}(r) = \mathsf{KEYGEN}(\mathsf{SK})$. This may be flat out impossible (in the case where the range of keygen is sparse), and at a minimum requires inverting keygen, again breaking the security of the NIKE scheme.

A third approach which does work is to combine the two. Namely, setting $\mathsf{KEYGEN}(\mathsf{SK}) = H_1(\mathsf{keygen}(H_0(\mathsf{SK})))$. Now both H_0, H_1 are random oracles that are simulated by the simulator. This actually gives indifferentiability: when the adversary queries $H_0(\mathsf{SK})$, the simulator will program $H_0(\mathsf{SK}) = r$ for a randomly chosen r. Then it will program $H_1(\mathsf{keygen}(r)) = \mathsf{KEYGEN}(\mathsf{SK})$ by querying KEYGEN. The only way a problem can arise is if the input $\mathsf{keygen}(r)$ was already programmed in H_1. All that we need to exclude such a possibility is that keygen is well-spread: that the distribution of outputs given a uniformly random input has high min-entropy. This follows easily from the security of the NIKE protocol.

The takeaway from the above discussion is that the inputs and outputs for a standard-model scheme must be processed by idealized objects; this is the only way that the simulator can get access to enough information in order to be indifferentiable.

Making $\mathsf{SHAREDKEY}$ *indifferentiable.* Next, we move to define $\mathsf{SHAREDKEY}$ in a way to make the joint oracles $(\mathsf{KEYGEN}, \mathsf{SHAREDKEY})$ indifferentiable from an ideal NIKE protocol.

Unfortunately, we immediately run into problems. We somehow need to design the shared-key algorithm $\mathsf{SHAREDKEY}$ to take as input $\mathsf{PK} = H_1(\mathsf{keygen}(H_0(\mathsf{SK})))$, as well as a secret key SK. It will output a shared key K. Importantly, we need to maintain the correctness requirement that $\mathsf{SHAREDKEY}(\mathsf{PK}_1, \mathsf{SK}_2) = \mathsf{SHAREDKEY}(\mathsf{PK}_2, \mathsf{SK}_1)$ whenever $\mathsf{PK}_1 = \mathsf{KEYGEN}(\mathsf{SK}_1)$ and $\mathsf{PK}_2 = \mathsf{KEYGEN}(\mathsf{SK}_2)$.

Guided by Impagliazzo and Rudich's [21] barrier, we cannot rely on the functionalities of the random oracles H_0, H_1 for this. Instead, we must use the functionality provided by $(\mathsf{keygen}, \mathsf{sharedkey})$. However, $\mathsf{sharedkey}$ expects output from keygen, and this value has been completely scrambled by the hash function H_1, which is un-invertible. Therefore, $\mathsf{SHAREDKEY}$ has no way to apply $\mathsf{sharedkey}$ in a meaningful way.

So we need some way to preserve the structure of the output keygen while still allowing for an indifferentiability proof. But at the same time, we cannot just expose the output of keygen in the clear, as explained above.

Our solution is to replace H_1 with a random *permutation* P such that *both* P and P^{-1} are publicly accessible (we discuss instantiating the random permutation below). We then have that

$$\mathsf{KEYGEN}(\mathsf{SK}) = P^{-1}(\mathsf{keygen}(H_0(\mathsf{SK})))$$

Then we can define $\mathsf{SHAREDKEY}(\mathsf{PK}, \mathsf{SK}) = \mathsf{enc}(P(\mathsf{PK}), H_0(\mathsf{SK}))$. Defining $\mathsf{SHAREDKEY}$ in this way achieves the desired correctness guarantee, which follows simply from the correctness of $(\mathsf{keygen}, \mathsf{SHAREDKEY})$.

However, by allowing the permutation P to be invertible, we have invalidated our indifferentiability proof above for KEYGEN. Suppose for example that keygen's outputs are easily distinguishable from random. Then an attacker can compute $\mathsf{PK} = \mathsf{KEYGEN}(\mathsf{SK})$, and then query P on either PK or a random string r. In the case of a random string, $P(r)$ will itself be essentially a random string. On the other hand, $P(\mathsf{PK})$ will be an output of keygen, and hence distinguishable from random. The problem is that the simulator defining P^{-1} only gets to see r and has no way to know whether r came from KEYGEN or was just a random string. Therefore, the attacker can fool the simulator, leading to a distinguishing attack.

To avoid this problem, we will assume the standard-model NIKE protocol has pseudorandom public keys. In this case, the simulator will *always* respond to a P using a fresh random output of keygen. In the case where the query to P was on a random r, the result will look random to the adversary. On the other hand, if the query was on a $\mathsf{PK} = \mathsf{KEYGEN}(\mathsf{SK})$, the simulator is ready to program any subsequent $H_0(\mathsf{SK})$ query to satisfy $P(\mathsf{PK}) = \mathsf{keygen}(H_0(\mathsf{SK}))$.

More problems still arise, similar to the problems above faced when trying to define $\mathsf{KEYGEN}(\mathsf{SK}) = \mathsf{keygen}(H(\mathsf{SK}))$. Namely, the adversary could first call the query $k = \mathsf{SHAREDKEY}(\mathsf{PK}', \mathsf{SK})$, which is the ideal world will give a random string. Then the adversary makes queries to P, H_0 and computes $\mathsf{sharedkey}(P(\mathsf{PK}'), H_0(\mathsf{SK}))$ itself. In the ideal world, the simulator must set $P(\mathsf{PK}') = r$ and $H_0(\mathsf{SK}) = s$ such that $\mathsf{sharedkey}(r, s) = k$. But this involves inverting $\mathsf{sharedkey}$ on k, which may be computationally infeasible. Worse yet, the adversary could do this for $\mathsf{PK}'_1, \dots, \mathsf{PK}'_\ell$ and $\mathsf{SK}_1, \dots, \mathsf{SK}_\ell$, obtaining ℓ^2 different random and independent $k_{i,j}$ values from $\mathsf{SHAREDKEY}$ by considering all possible $\mathsf{PK}'_i, \mathsf{SK}_j$ pairs. The simulator then needs to somehow find $r_1, \dots, r_\ell, s_1, \dots, s_\ell$ such that $\mathsf{sharedkey}(r_i, s_j) = k_{i,j}$, where $k_{i,j}$ are each random independent strings. This is clearly impossible for large enough ℓ, since it would allow for compressing an $O(\ell^2)$-bit random string into $O(\ell)$ bits.

Our solution to this problem is to apply one more hash function, this time to the output of $\mathsf{sharedkey}$: $\mathsf{SHAREDKEY}(\mathsf{PK}_0, \mathsf{SK}_1) = H_1(\mathsf{sharedkey}(P(\mathsf{PK}_0), H_0(\mathsf{SK}_1)))$.

How does this get around the problem above? Now, all we need is that $\mathsf{sharedkey}(r_i, s_j)$ are all distinct for different i, j pairs, which follows with high probability from the security of the NIKE scheme. Then we can simply program $H_1(\mathsf{sharedkey}(r_i, s_j)) = k_{i,j}$.

Our final construction is actually a slight tweak:

$$\mathsf{SHAREDKEY}(\mathsf{PK}_0, \mathsf{SK}_1) = H_1(\{\mathsf{PK}_0, \mathsf{PK}_1\}, \mathsf{sharedkey}(P(\mathsf{PK}_0), H_0(\mathsf{SK}_1)))$$

where $\mathsf{PK}_1 = \mathsf{KEYGEN}(\mathsf{SK}_1) = P^{-1}(\mathsf{keygen}(H_0(\mathsf{SK}_1)))$. Here, $\{\mathsf{PK}_0, \mathsf{PK}_1\}$ means the unordered set containing PK_0 and PK_1, so that $\{\mathsf{PK}_0, \mathsf{PK}_1\} = \{\mathsf{PK}_1, \mathsf{PK}_0\}$. Including $\mathsf{PK}_0, \mathsf{PK}_1$ in the hash is needed in situations where it is possible, given s to compute r_0, r_1 such that $\mathsf{sharedkey}(r_0, s) = \mathsf{sharedkey}(r_1, s)$. In this case, an attacker can query $H_0(\mathsf{SK})$ to get s, and then compute r_0, r_1. If we did not include $\mathsf{PK}_0, \mathsf{PK}_1$ in the hash, $\mathsf{SHAREDKEY}(P^{-1}(r_0), \mathsf{SK}) = \mathsf{SHAREDKEY}(P^{-1}(r_1), \mathsf{SK})$. However, this would almost never happen in the ideal world. By including $\mathsf{PK}_0, \mathsf{PK}_1$ in the hash, we ensure that it is infeasible to find collisions in $\mathsf{SHAREDKEY}$, even if it is possible to find collisions in $\mathsf{sharedkey}$.

One last attack strategy: the attacker could first query $\mathsf{PK}_0 = \mathsf{KEYGEN}(\mathsf{SK}_0), \mathsf{PK}_1 = \mathsf{KEYGEN}(\mathsf{SK}_1), k = \mathsf{SHAREDKEY}(\mathsf{PK}_0, \mathsf{SK}_1)$. Then, it could query $r_0 = P(\mathsf{PK}_0), r_1 = P(\mathsf{PK}_1)$. Finally, it could treat r_0, r_1 as the messages in the standard-model NIKE protocol, and guess the shared key t for the protocol. Then it could query H_2 on t (and $\{\mathsf{PK}_0, \mathsf{PK}_1\}$). In the real world, the result $H_2(\{\mathsf{PK}_0, \mathsf{PK}_1\}, t)$ would be equal to k, so the simulator in the ideal world needs to be able to set $H_2(\{\mathsf{PK}_0, \mathsf{PK}_1\}, t) = k$. At this point, the simulator has $\mathsf{PK}_0, \mathsf{PK}_1, t$. But the simulator has no knowledge of SK_0 or SK_1. Therefore it has no way of guessing the correct input to $\mathsf{SHAREDKEY}$ to obtain k, as doing so requires recovering either SK_0 or SK_1 by inverting the random injection KEYGEN. Fortunately, this is a non-attack by the security of the underlying standard-model NIKE protocol. Indeed, guessing the shared key t from the user's messages r_0, r_1 is exactly what NIKE security protects against.

While we have protected against certain natural attacks, we need to argue indifferentiability against all possible attacks. To do so we use a careful simulation strategy for H_0, H_1, P, P^{-1}, and prove indifferentiability through a careful sequence of hybrids. In essence, each hybrid corresponds roughly to one of the attack strategies discussed above, and our proof shows that these attacks do not work, demonstrating the indistinguishability of the hybrids.

Constructing P, P^{-1}. Our random permutation P, P^{-1} can easily be instantiated using the ideal cipher model in the setting where the key space contains on a single element. We note that indifferentiable ideal ciphers can be constructed from random oracles [13].

Therefore, all we need for our construction is three random oracles. Multiple random oracles can easily be built from a single random oracle by prefixing the oracle index to the input. Finally, an indifferentiable random oracle of any domain/range can be constructed from a fixed-size random oracle [11].

The role of P *in the proof.* It is natural to wonder what role P plays in the actual security of our scheme. After all, since P^{-1} is publicly invertible using P, the adversary can easily undo the application of P^{-1} to the output of KEYGEN. So it may seem that P is a superfluous artifact of the proof.

There are multiple ways to address this question. One answer is that without P, there would be no way to have a computationally efficient simulator as discussed above. One could consider an inefficient simulator, but this would correspond to a weaker notion of indifferentiability. This notion of indifferentiability would be useless when composing with protocols that have computational rather than statistical security. What's more, we would actually be unable to prove even this weaker form. Indeed, our proof crucially relies on the computational security of the standard-model NIKE protocol. Since the inefficient simulator would essentially have to break the security of the NIKE protocol, it would be impossible to carry out the proof.

A higher-level answer is that by including P — which is under full control of the simulator — the simulator gets to learn extra information about what values the adversary is interested in. In particular, in order to relate the ideal oracles to the standard-model scheme, the adversary must always send a query to the simulator. This extra information provided by making such queries is exactly what the simulator needs for the proof to go through. This is a common phenomenon in random-oracle proofs, where hashing sometimes has no obvious role except to provide a reduction/simulator with necessary information.

Yet another answer is that, if P is omitted, the scheme is actually insecure in some settings. For example, an ideal NIKE satisfies the property that an adversary, given Alice's secret key and *half* of Bob's public key, cannot compute the shared key between Alice and Bob. Now, consider the case where the standard model NIKE does *not* satisfy this requirement. Then if we do not include P, our construction does not satisfy the requirement either. Instead, by including P, an adversary who gets half of Bob's ideal public key cannot invert the permutation to recover *any* information about the corresponding standard-model public key. It then follows that the adversary cannot guess the shared key.

The takeaway is that, only by including P so that we achieve full indifferentiability can we ensure that such settings do not arise.

1.5 Extending to Other Idealized Cryptosystems

We now turn our attention to extending the above results to other cryptosystems. First, we use our ideal NIKE scheme to construct ideal public key encryption (PKE). Note that ideal public key encryption is in particular CCA secure, whereas the standard way to turn a NIKE scheme into a PKE scheme is never CCA secure. In order to make the scheme CCA secure, a natural starting point is the Fujisaki-Okamoto (FO) transform [20]. While this transformation applied to our ideal NIKE certainly achieves CCA security, it unfortunately is not indifferentiable when applied to our NIKE. The reasons are several-fold, and should come as no surprise given that FO was never designed to achieve indifferentiability.

For starters, recall from our NIKE discussion that all inputs and outputs of the algorithms need to be passed through ideal objects under the simulator's control. In the FO transformation, this is not the case. Another reason why FO does not give indifferentiability is that the FO transform allows for encryption randomness to be recovered during decryption; in fact, this is a crucial feature of the CCA security proof. On the other hand, such encryption schemes cannot be ideal, since ideal encryption schemes guarantee that the encryption randomness is hidden even to the decrypter[5].

To overcome these issues, we first show a careful transformation from our ideal NIKE into ideal *deterministic* public key encryption (DPKE). By focusing first on DPKE, we side-step the randomness issue. Our transformation is inspired by the FO transform, but in order to ensure that all inputs/outputs are passed through oracles under the simulator's control, we employ our random permutation trick again.

Finally, we turn to convert ideal DPKE into standard PKE. The usual conversion (simply including the encryption randomness as part of the message) does not suffice, again because the usual conversion allows the decrypter to recover the encryption randomness. We instead essentially hide the randomness by hashing with a random oracle. This, however, requires care in order to enable a complete indifferentiability proof.

Ideal Signatures. Finally, we investigate constructing ideal signatures. While in the standard model signatures can in principle be built from one-way functions and therefore random oracles, we observe that the situation for ideal signatures is much more challenging. For example, an ideal signature scheme will be *unique*, meaning for any message/public key, only a single signature will verify. On the other hand, constructing unique signatures even under standard security notions is difficult, and the only known constructions require strong number-theoretic tools such as bilinear maps.

We instead assume a building block as a standard-model signature scheme with unique signatures, as well as some other mild security properties which can be easily instantiated using bilinear maps. We show that such a scheme can, in fact, be turned into ideal signatures using similar ideas to the above.

1.6 Instantiations

Our NIKE schemes require a standard-model NIKE. Unfortunately, we cannot use a truly arbitrary standard model NIKE, as we need that keygen has (pseudo)random outputs in $\{0,1\}^n$. As such, we need to make sure such a scheme can be instantiated. Our other results similarly require standard-model schemes where various outputs of the schemes are pseudorandom bit strings.

Such a requirement is slightly non-trivial. Many number-theoretic constructions have public keys that are elements in $\mathbb{Z}_q^k$ for some modulus k; even if the public keys are pseudorandom in these sets, there may be no way to represent a random element of $\mathbb{Z}_q^k$ as a random bit string, since $q^k = |\mathbb{Z}_q^k|$ may be far from a power of 2.

[5] One can define a different idealization of PKE where the ideal decryption functionality *does* output the encryption randomness. However, this stronger functionality corresponds to weaker security guarantees

However, it will usually be easy to map such public keys to random strings in $\{0,1\}^n$ for some integer n. For example, in the case $k=1$, suppose we are given a (pseudo)random element $\mathsf{pk} \in \mathbb{Z}_q$. Let n be some integer such that $n \geq \lambda + \log_2 q$ for a security parameter λ. Let $t = \lfloor 2^n/q \rfloor$ be the largest integer such that $tq \leq 2^n$. Then we can extend pk into a random element pk' in $\mathbb{Z}_{tq}$ by setting $\mathsf{pk}' = \mathsf{pk} + aq$, where a is a random integer in $\mathbb{Z}_t$. Finally, we note that a random integer in $\mathbb{Z}_{tq}$ is distributed exponentially close (in λ) to a random integer in $\mathbb{Z}_{2^n}$.

We can similarly handle the case $k > 1$ by bijecting public keys into $\mathbb{Z}_{q^k}$ in the standard way. Such conversions can be applied to Diffie-Hellman key agreement as well as typical lattice-based NIKE schemes. The result that we attain our NIKE/PKE results under either CDH or LWE, whereas our signature scheme requires CDH in bilinear map groups.

We note that one of the cases we do not know how to handle are schemes based on factoring or RSA, as the public key would be an RSA composite $p \times q$ for random large primes p and q; this is clearly distinguishable from a random string using primality testing. On the other hand, we know of no way to bijectively map such numbers into random bit strings, without factoring them (and hence breaking the scheme). In fact, doing so without factoring would seem to allow for *obliviously* sampling large RSA composites by choosing a random string, and then applying the inverse map. Oblivious sampling of RSA composites is a major open question.

2 Background

Notation. Throughout this paper, $\lambda \in \mathbb{N}$ denotes the security parameter. We let $\mathbb{N}$ be the set of non-negative integers, including zero and $\{0,1\}^*$ denote the set of all finite-length bit strings, including the empty string ϵ ($\{0,1\}^0 = \epsilon$). For two bit strings, X and Y, $X||Y$ denotes string concatenation and (X, Y) denotes a uniquely decodable encoding of X and Y. The length of a string X is denoted by $|X|$.

For a finite set $\mathcal{S}$, we denote $s \leftarrow \mathcal{S}$ the process of sampling s uniformly from $\mathcal{S}$. For a probabilistic algorithm A, we denote $y \leftarrow A(x; R)$ the process of running A on inputs x and randomness R, and assigning y the result. We let $\mathcal{R}_A$ denote the randomness space of A; we require $\mathcal{R}_A$ to be the form $\mathcal{R}_A = \{0,1\}^r$. We write $y \leftarrow A(x)$ for $y \leftarrow A(x, R)$ with uniformly chosen $R \in \mathcal{R}_A$, and we write $y_1, \ldots, y_m \leftarrow A(x)$ for $y_1 \leftarrow A(x), \ldots, y_m \leftarrow A(x)$ with fresh randomness in each execution. If A's running time is polynomial in λ, then A is called probabilistic polynomial-time (PPT). We say a function $\mu(n)$ is negligible if $\mu \in o(n^{-\omega(1)})$, and is non-negligible otherwise. We let $\mathsf{negl}(n)$ denote an arbitrary negligible function. If we say some $p(n)$ is poly, we mean that there is some polynomial q such that for all sufficiently large n, $p(n) \leq q(n)$. We say a function $\rho(n)$ is noticeable if the inverse $1/\rho(n)$ is poly. We use boldface to denote vector, i.e. $\boldsymbol{m}$; we denote $\boldsymbol{m}_i$ as the i-th component of $\boldsymbol{m}$ and $|\boldsymbol{m}|$ as the length of $\boldsymbol{m}$.

Games. An n-adversary game $\mathcal{G}$ is a Turing machine $\mathcal{G}^{\Sigma, \mathcal{A}_1, \ldots, \mathcal{A}_n}$ where Σ is a system (or functionality) and $\mathcal{A}_i$ are adversarial procedures that can keep full local state but may only communicate with each other through $\mathcal{G}$. We say a n-adversary game $\mathcal{G}_n$ is reducible to an m-adversary game if there is a $\mathcal{G}_m$ such that for any $(\mathcal{A}_1, \ldots, \mathcal{A}_n)$ there are $(\mathcal{A}'_1, \ldots, \mathcal{A}'_m)$ such that for all Σ we have that $\mathcal{G}_n^{\Sigma, \mathcal{A}_1, \ldots, \mathcal{A}_n} = \mathcal{G}_m^{\Sigma, \mathcal{A}'_1, \ldots, \mathcal{A}'_m}$. Two games are equivalent if they are reducible in both directios. An n-adversary game is called n-stage [25] if it is not equivalent to any m-adversary game with $m < n$. Any single-stage game $\mathcal{G}_{\Sigma, \mathcal{A}}$ can be also written as $\bar{\mathcal{A}}^{\bar{\mathcal{G}}^{\Sigma}}$ for some oracle machine $\bar{\mathcal{G}}$ and a class of adversarial procedures $\bar{\mathcal{A}}$ compatible with a modified syntax in which the game is called as an oracle.

2.1 Public Key Primitives

In this part, we recall the definitions of the public key primitives that we consider in our work.

Non-Interactive Key Exchange (NIKE). NIKE is a cryptographic primitive which enables two parties, who know the public keys of each other, to agree on a symmetric shared key without requiring any interaction. The canonical example of a NIKE scheme can be found in the seminal paper by Diffie and Hellman [14]. Following [10] and [19], we formally define NIKE in the public key setting to be a collection of two algorithms: $\mathsf{NIKE.keygen}$ and $\mathsf{NIKE.sharedkey}$ together with a shared key space SHK.

- $\mathsf{NIKE.keygen}$: On inputs a secret key sk, outputs the corresponding public key pk;
- $\mathsf{NIKE.sharedkey}$ On inputs a public key $\mathsf{pk_0}$ and a secret key $\mathsf{sk_1}$, outputs a shared key $\mathsf{shk} \in \mathsf{SHK}$.

For correctness, we require that, for any two key pairs $(\mathsf{pk_0}, \mathsf{sk_0}), (\mathsf{pk_1}, \mathsf{sk_1})$, algorithm $\mathsf{NIKE.sharedkey}$ satisfies:

$$\mathsf{NIKE.sharedkey}(\mathsf{pk_0}, \mathsf{sk_1}) = \mathsf{NIKE.sharedkey}(\mathsf{pk_1}, \mathsf{sk_0}).$$

Public Key Encryption (PKE) [14]. A public-key encryption scheme consists of three algorithms: $\mathsf{PKE.keygen}, \mathsf{PKE.enc}, \mathsf{PKE.dec}$ together with a message space $\mathcal{M}$. The key-generation algorithm $\mathsf{PKE.keygen}$ produces a secret key and a corresponding public key; $\mathsf{PKE.enc}$ uses the public key for mapping plaintexts into ciphertexts, and $\mathsf{PKE.dec}$ recovers plaintexts from ciphertexts by using the secret key. Formally,

- $\mathsf{PKE.keygen}$ On inputs 1^λ, outputs a public/secret key pair $(\mathsf{pk}, \mathsf{sk})$;
- $\mathsf{PKE.enc}$ On inputs a public key pk and a message $\mathsf{m} \in M$, outputs a ciphertext $c = \mathsf{PKE.enc}(\mathsf{pk}, \mathsf{m})$;
- $\mathsf{PKE.dec}$ On inputs a ciphertext c and a secret key, outputs either a plaintext m or $\perp$.

For correctness, we require that, for any key pair $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{PKE.keygen}(1^\lambda)$ and $\mathsf{m} \in \mathcal{M}$, the scheme satisfies:

$$\mathsf{PKE.dec}(\mathsf{sk}, \mathsf{PKE.enc}(\mathsf{pk}, \mathsf{m})) = \mathsf{m}.$$

Digital Signature [26]. A digital signature scheme consists of three algorithms: $\mathsf{Sig.keygen}$, $\mathsf{Sig.sign}, \mathsf{Sig.ver}$ along with a message space $\mathcal{M}$. The key-generation algorithm $\mathsf{Sig.keygen}$ produces a sign key and a corresponding verification key; $\mathsf{Sig.sign}$ uses the sign key for mapping messages into signatures; and $\mathsf{Sig.ver}$ checks the validity of the signature. Formally,

- $\mathsf{Sig.keygen}$ On inputs 1^λ, outputs a sign/verification key pair $(\mathsf{sk}, \mathsf{vk})$;
- $\mathsf{Sig.sign}$ On inputs a sign key and a message $\mathsf{m} \in \mathcal{M}$, outputs a signature $\sigma = \mathsf{Sig.sign}(\mathsf{sk}, \mathsf{m})$
- $\mathsf{Sig.ver}$ On inputs a signature σ, a message and a verification key vk, outputs either 1 or 0.

For correctness, we require that, for any key pair $(\mathsf{sk}, \mathsf{vk}) \leftarrow \mathsf{Sig.keygen}(1^\lambda)$ and $\mathsf{m} \leftarrow \mathcal{M}$, the signature scheme satisfies:

$$\mathsf{Sig.ver}(\mathsf{Sig.sign}(\mathsf{sk}, \mathsf{m}), \mathsf{m}, \mathsf{vk}) = 1$$

2.2 Ideal Objects

In the section, we recall three lower level ideal objects.

Random Oracle Model (ROM) [4, 18]. In the random oracle model, one assumes that some hash function is replaced by a publicly accessible random function (the random oracle). This means that the adversary cannot compute the result of the hash function by himself: it must query the random oracle. More concretely, random oracle model is a hash function $H : \{0,1\}^* \to \{0,1\}^n$ such that 1) each bit of $H(x)$ is random; 2) the output is uniform and independent.

Ideal-Cipher Model (ICM) [28, 5].The Ideal Cipher Model is another idealized model of computation, similar to the ROM. Instead of having a publicly accessible random function, one has a publicly accessible random block cipher (or ideal cipher). This is a block cipher with a k-bit key and a n-bit input/output, that is chosen uniformly at random among all block ciphers of this form; this is equivalent to having a family of 2^k independent random permutations. Specifically, an ideal cipher model is a keyed permutation $P : \{0,1\}^k \times \{0,1\}^n \to \{0,1\}^n$ such that 1)for any fixed key k, P_k is a random permutation (independent of k); 2) adversaries interacting with an ideal-cipher oracle are typically given access to both the cipher and its inverse.

Random Functions. Let $\mathcal{X}$ and $\mathcal{Y}$ be two sets, we define $\mathcal{F}[\mathcal{X} \to \mathcal{Y}]$ be the set of all functions that map $\mathcal{X}$ to $\mathcal{Y}$. We say F is a random function, if F is sampled uniformly from $\mathcal{F}[\mathcal{X} \to \mathcal{Y}]$. Granting oracle accesses to F to all parties (honest or adversarial) results in an ideal model of computation.

Random Injections. Similarly, let X and Y be two sets, we define $\mathrm{Inj}[X \to Y]$ be the set of all functions that map X to Y. We say F is a random injection if F is sampled uniformly from $\mathrm{Inj}[X \to Y]$.

Lazy samplers. Various ideal objects, for instance, random oracle, often appear as algorithmic procedures that lazily sample function at each point. Moreover it is well known that lazy samplers for random functions and ideal ciphers with arbitrary domain and range exist. In addition, in the case of random injections, Rogaway and Shrimpton [27] give a procedure to sample injections properly, for any domain and range.

2.3 Indifferentiability

The indifferentiability framework of Maurer, Renner and Holenstein(MRH) [23] formalized a set of necessary and sufficient conditions for one system to securely replace another in a wide class of environments. This framework has been successfully used to justify the structural soundness of a number of cryptographic constructions, such as hash functions [11, 15], blockciphers [1, 13, 16], domain extenders [12] and authenticated encryption with associated data [2]. In this section, we recall the concept of indifferentiability of systems.

A random system or functionality $\Sigma := (\Sigma.\mathrm{hon}, \Sigma.\mathrm{adv})$ is accessible via two interfaces $\Sigma.\mathrm{hon}$ and $\Sigma.\mathrm{adv}$. Here, $\Sigma.\mathrm{hon}$ provides a public interface through which the system can be accessed, $\Sigma.\mathrm{adv}$ corresponds to a (possibly extended) interface that models adversarial access to the inner working parts of the system, which might be exploited during an attack on constructions. A system typically implements some ideal objects $\mathcal{F}$, or it is a construction $C^{\mathcal{F}'}$, which applies some underlying (lower-level) ideal object $\mathcal{F}'$.

Definition 1. (**Indifferentiability** [23].) *Let* Σ_1 *and* Σ_2 *be two systems and* $\mathcal{S}$ *be an algorithm called the simulator. The (strong) indifferentiability advantage of a (possibly unbounded) differentiator* $\mathcal{D}$ *against* (Σ_1, Σ_2) *with respect* $\mathcal{S}$ *is*

$$\mathrm{Adv}^{\mathsf{indif}}_{\Sigma_1,\Sigma_2,\mathcal{S},\mathcal{D}}(1^\lambda) := \Pr[\mathrm{Real}_{\Sigma_1,\mathcal{D}}] - \Pr[\mathrm{Ideal}_{\Sigma_2,\mathcal{S},\mathcal{D}}],$$

where games $\mathrm{Real}_{\Sigma_1,\mathcal{D}}$ *and* $\mathrm{Ideal}_{\Sigma_2,\mathcal{S},\mathcal{D}}$ *are defined in Figure 1 . We say* Σ_1 *is indifferentiable from* Σ_2 *if for all probabilistic polynomial time (unbounded) adversares, there exists an "efficient"* $\mathcal{S}$*, the advantage above is negligible.*

GAME $\mathrm{Real}_{\Sigma_1,\mathcal{D}}$:	PROC.CONST(X)	GAME $\mathrm{Ideal}_{\Sigma_2,\mathcal{D}}$:	PROC.CONST(X)
$b \xleftarrow{\$} \mathcal{D}^{\text{CONST},\text{PRIM}}$	Return Σ_1.hon(X)	$b \xleftarrow{\$} \mathcal{D}^{\text{CONST},\text{PRIM}}$	Return Σ_2.hon(X)
Return b		Return b	
	PROC.PRIM(X)		PROC.PRIM(X)
	Return Σ_1.adv(X)		Return $\mathcal{S}^{\Sigma_2.\mathrm{adv}(\cdot)}(X)$

Fig. 1: Indifferentiability of Σ_1 and Σ_2, where $\mathcal{S}$ is the simulator and $\mathcal{D}$ is the adversary.

In the rest of the paper, we consider the definition above to two systems with interfaces:

$$(\Sigma_1.\mathrm{hon}(\mathrm{X}), \Sigma_1.\mathrm{adv}(\mathrm{x})) := (\mathrm{C}^{\mathcal{F}_1}(\mathrm{X}), \mathcal{F}_1(\mathrm{x})); (\Sigma_2.\mathrm{hon}(\mathrm{X}), \Sigma_2.\mathrm{adv}(\mathrm{x})) := (\mathcal{F}_2(\mathrm{X}), \mathcal{F}_2(\mathrm{x})),$$

where $\mathcal{F}_1$ and $\mathcal{F}_2$ are two ideal cryptographic objects sampled from their distributions and $\mathrm{C}^{\mathcal{F}_1}$ is an encoding of $\mathcal{F}_2$ by calling $\mathcal{F}_1$.

In [23], MRH proves the following composition theorem for indifferentiable systems. For ease, we give a game-based formulation by Ristenpart, Shacham and Shrimpton(RSS) [25].

Theorem 2. (**Indifferentiability Composition** [25].) *Let* $\Sigma_1 := (\mathrm{C}^{\mathcal{F}_1}, \mathcal{F}_1)$ *and* $\Sigma_2 := (\mathcal{F}_2, \mathcal{F}_2)$ *be two indifferentiable systems with simulator* $\mathcal{S}$*. Let* $\mathbf{G}$ *be a single-stage game. Then for any adversary* $\mathcal{A}$ *there exists an adversary* $\mathcal{B}$ *and a differentiator* $\mathcal{D}$ *such that*

$$\Pr[\mathbf{G}^{\mathrm{C}^{\mathcal{F}_1},\mathcal{A}^{\mathcal{F}_1}}] \le \Pr[\mathbf{G}^{\mathcal{F}_2,\mathcal{B}^{\mathcal{F}_2}}] + \mathrm{Adv}^{\mathsf{indif}}_{\Sigma_1,\Sigma_2,\mathcal{S},\mathcal{D}}$$

Proof. It's trivial to note that

$$\Pr[\mathbf{G}^{\mathrm{C}^{\mathcal{F}_1},\mathcal{A}^{\mathcal{F}_1}}] \approx \Pr[\mathbf{G}^{\mathcal{F}_2,\mathcal{A}^{\mathcal{S}^{\mathcal{F}_2}}}] = \Pr[\mathbf{G}^{\mathcal{F}_2,\mathcal{B}^{\mathcal{F}_2}}],$$

where the first transition follows from indifferentiability of $\mathbf{G}$, and the second one by viewing $\mathcal{B} = \mathcal{A}^{\mathcal{S}}$. □

In [25], RSS show that the composition above does not necessarily extend to multi-stage games because the simulator has to keep the *local* state in order to guarantee consistency. However, if adding some restrictions, one could rewrite some multi-stage games as equivalent single-stage games. In fact, any n-adversary game where only one adversary can call the primitive directly and the rest call it indirectly via the construction can be written as a single-stage as the game itself has access to the construction. Barbosa and Farshim [2] generalize a result for related-key security [17] and formalize the observation in the following theorem.

Theorem 3. (**Multi-stage Game Composition** [2].) *Let* $\Sigma_1 := (\mathrm{C}^{\mathcal{F}_1}, \mathcal{F}_1)$ *and* $\Sigma_2 := (\mathcal{F}_2, \mathcal{F}_2)$ *be two indifferentiable systems with simulator* $\mathcal{S}$*. Let* $\boldsymbol{G}$ *be an* n*-adversary game and* $\mathcal{A} := (\mathcal{A}_1, \ldots, \mathcal{A}_n)$ *be a* n*-tuple of adversaries where* $\mathcal{A}_1$ *can access* F_1 *but* $\mathcal{A}_i$ *(i¿1) can only access* $\mathrm{C}^{\mathcal{F}_1}$*. Then there is an* n*-adversary* $\mathcal{B}$ *and a differentiator* $\mathcal{D}$ *such that*

$$\Pr[\boldsymbol{G}^{\mathrm{C}^{\mathcal{F}_1}, \mathcal{A}_1^{\mathcal{F}_1}, \mathcal{A}_2^{\mathrm{C}^{\mathcal{F}_1}}, \ldots, \mathcal{A}_{\mathrm{n}}^{\mathrm{C}^{\mathcal{F}_1}}}] = \Pr[\boldsymbol{G}^{\mathcal{F}_2, \mathcal{B}_1^{\mathcal{F}_2}, \ldots, \mathcal{B}_n^{\mathcal{F}_2}}] + \mathrm{Adv}^{\mathsf{indif}}_{\Sigma_1, \Sigma_2, \mathcal{S}, \mathcal{D}}.$$

Remark. Barbosa and Farshim [2] give a strong motivation for the restriction imposed on the class of games above. To our best of knowledge, the related-key attack (key-dependent message attack) game is not known to be equivalent to a single-stage game. Thus, in order to prove one system, for instance Σ_1, is related-key attack secure, it's not yet sufficient to prove 1) there is another system, say Σ_2, such that Σ_1 is indifferentiable from Σ_2; 2) Σ_2 is related-key attack secure. However, if we restrict that, adversaries only access the related-key deriving functions via the construction only, then the proof follows. Therefore, from a practical point of view, composition extends well beyond 1-adversary games.

3 Indifferentiable NIKE

In this section, we propose the notion of "ideal NIKE" and then build an indifferentiable non-interactive key exchange scheme from simpler ideal primitives and a standard-model NIKE scheme.

3.1 What is Ideal NIKE?

In this part we give the rigorous description of ideal NIKE, formally:

Definition 4. (**Ideal NIKE.**) *Let* $\mathcal{X}, \mathcal{Y}$*,* $\mathcal{Z}$ *be three sets such that* $|\mathcal{X}|, |\mathcal{Y}|, |\mathcal{Z}| \geq 2^{\omega(\log \lambda)}$*,* $|\mathcal{X}| \leq |\mathcal{Y}|$ *and* $|\mathcal{X}| \times |\mathcal{Y}| \leq |\mathcal{Z}|$*. We denote* $\mathcal{F}[\mathcal{X} \to \mathcal{Y}]$ *as the set of all injections that map* $\mathcal{X}$ *to* $\mathcal{Y}$ *and* $\mathcal{G}[\mathcal{X} \times \mathcal{Y} \to \mathcal{Z}]$ *as the set of the functions that map* $\mathcal{X} \times \mathcal{Y}$ *to* $\mathcal{Z}$*. We define* $\mathcal{T}$ *as the set of all function pairs* (F, G) *such that: 1)*$F \in \mathcal{F}, G \in \mathcal{G}$*; 2)* $\forall x, y \in \mathcal{X}, G(x, F(y)) = G(y, F(x))$*; 3)* $G(x_1, y_1) = G(x_2, y_2) \Rightarrow (x_1 = x_2, y_1 = y_2) \cup (y_1 = F(x_1), y_2 = F(x_2))$*.*

We say that a NIKE scheme $\Pi_{\mathsf{NIKE}} = (\mathsf{NIKE.KEYGGEN}, \mathsf{NIKE.SHAREDKEY})$*, associated with secret key space* $\mathcal{X}$*, public key space* $\mathcal{Y}$ *and shared key space* $\mathcal{Z}$*, is an ideal NIKE if* $(\mathsf{NIKE.KEYGEN}, \mathsf{NIKE.SHAREDKEY})$ *is sampled from* $\mathcal{T}$ *uniformly.*

It's trivial to note that, due to the information-theoretic argument, an ideal NIKE achieves related-key attack secure, leakage-resilient and so forth. Next, we show how we construct an indifferentiable NIKE scheme from simpler primitives.

3.2 Construction

In this section, we build an indifferentiable NIKE scheme from simpler ideal primitives (namely random oracles and ideal ciphers) along with a standard-model (that is, *non-ideal*) NIKE scheme.

Building Blocks. Our scheme will consist of several building blocks:

- A standard-model NIKE scheme $\Pi_{\mathsf{SM-NIKE}} = (\mathsf{keygen}, \mathsf{sharedkey})$ with secret key space $\mathcal{X}$, public key space $\mathcal{Y}$, and shared key space $\mathcal{Z}$;

- $H_0 := \{0,1\}^* \to \mathcal{X}$ is a random oracle whose co-domain matches the secret key space of Π.
- $H_1 := \{0,1\}^* \to \mathcal{W}$ is a random oracle, where $|\mathcal{X}| \times |\mathcal{Y}| \leq |\mathcal{W}|$;
- $P := \mathcal{Y} \to \mathcal{Y}$ is a random permutation on the public key space of Π, and P^{-1} is P's inverse.

Construction. Now we are ready to build an indifferentiable NIKE scheme, denoted as $\Pi_{\mathsf{NIKE}} = (\mathsf{NIKE.KEYGEN}, \mathsf{NIKE.SHAREDKEY})$, from the building blocks above. Formally,

- $\mathsf{NIKE.KEYGEN}(\mathsf{SK})$: On inputs a secret key SK, the algorithm runs $\mathsf{keygen}(H_0(\mathsf{SK}))$, and outputs the public key $\mathsf{PK} = P^{-1}(\mathsf{keygen}(H_0(\mathsf{SK})))$;
- $\mathsf{NIKE.SHAREDKEY}(\mathsf{PK}_1, \mathsf{SK}_2)$: On inputs a public key PK_1 and a secret key SK_2, the algorithm computes $\mathsf{sharedkey}(P(\mathsf{PK}_1), H_0(\mathsf{SK}_2))$ and $\mathsf{PK}_2 = \mathsf{NIKE.KEYGEN}(\mathsf{SK}_2)$. If $\mathsf{PK}_1 \leq \mathsf{PK}_2$, then it outputs the shared key as

$$\mathsf{SHK} = H_1(\mathsf{PK}_1, \mathsf{PK}_2, \mathsf{sharedkey}(P(\mathsf{PK}_1), H_0(\mathsf{SK}_2))),$$

  eles, it outputs

$$\mathsf{SHK} = H_1(\mathsf{PK}_2, \mathsf{PK}_1, \mathsf{sharedkey}(P(\mathsf{PK}_1), H_0(\mathsf{SK}_2))).$$

Correctness of the scheme easily follows, and what's more interesting is its indifferentiability. Next, we prove our scheme is indifferentiable from an ideal NIKE. Before that, we firstly specify the security properties of the standard-model NIKE.

Property 1. UNPREDICTABLE SHARED KEY. We say the shared key, for a NIKE scheme, is unpredictable, if for any PPT adversary $\mathcal{A}$, the advantage

$$\mathrm{Adv}_{\mathcal{A}} := \Pr[\mathcal{A}(\mathsf{pk}_1, \mathsf{pk}_2) = \mathsf{sharedkey}(\mathsf{pk}_1, \mathsf{sk}_2)] \leq \mathsf{negl}(\lambda)$$

where $\mathsf{pk}_i = \mathsf{keygen}(\mathsf{sk}_i), \mathsf{sk}_i \leftarrow \mathcal{X}$.

Property 2. PSEUDORANDOM SHARED KEY. We say the shared key, for a NIKE scheme, is pseudorandom, if for any PPT adversary $\mathcal{A}$, the advantage

$$\mathrm{Adv}_{\mathcal{A}} := |\Pr[\mathcal{A}^{\mathsf{sk}^*}(\mathsf{sharedkey}(\mathsf{pk}, \mathsf{sk}^*))] - \Pr[\mathcal{A}(R)]| \leq \mathsf{negl}(\lambda)$$

where sk^* is chosen by the adversary and $\mathsf{pk} \leftarrow \mathcal{Y}, R \leftarrow \mathcal{Z}$.

Property 3. PSEUDORANDOM PUBLIC KEY. We say the public key, for a NIKE scheme, is pseudorandom, if for any PPT adversary $\mathcal{A}$, the advantage

$$\mathrm{Adv}_{\mathcal{A}} := |\Pr[\mathcal{A}(\mathsf{keygen}(r))] - \Pr[\mathcal{A}(R)]| \leq \mathsf{negl}(\lambda)$$

where $r \leftarrow \mathcal{X}, R \leftarrow \mathcal{Y}$.

We say a NIKE scheme is Good if it satisfies the three properties above. Moreover, note that constructions of random permutations typically operate on bit strings, meaning $\mathcal{Y} = \{0,1\}^n$ for some polynomial n. This means our standard-model NIKE must have public keys that are pseudorandom bit strings. Next, we prove the following theorem.

Theorem 5. *(**(Indifferentiable NIKE).**) Π_{NIKE} is indifferentiable from an ideal NIKE if $\Pi_{\mathsf{SM-NIKE}}$ is* Good.

Proof. According to the definition of indifferentiability, we immediately observe that the adversary has two honest interfaces ($\mathsf{NIKE.KEYGEN}$, $\mathsf{NIKE.SHAREDKEY}$) (below we will denote ($\mathsf{NKG}, \mathsf{NSK}$) for ease) and four adversarial interfaces (H_0, H_1, P,P^{-1}). Therefore, we need to build an efficient simulator $\mathcal{S}$ that can simulate the four adversarial interfaces H_0, H_1, P and P^{-1} properly, which means, for any PPT differentiator $\mathcal{D}$, the view of $\mathcal{D}$ in the real game is computationally close to the view in the ideal game. To do so, we will go through with a sequence of hybrid games. Before the description of the games, we first specify some parameters and events:

- there are six types of query such as: $(x, H_0), (y_1, y_2, z, H_1), (y, P), (y, P^{-1}), (x, \mathsf{NKG})$, (y, x, NSK) where $x \leftarrow \mathcal{X}, y, y_1, y_2 \leftarrow \mathcal{Y}, z \leftarrow \mathcal{Z}$;
- adversary makes at most q queries to the system, where $q = \mathsf{poly}(\lambda)$;
- the real oracles used in the construction are $\tilde{H}_0, \tilde{H}_1, \tilde{P}, \tilde{P}^{-1}$;
- the advantage of *unpredictable shared key* is bounded by ϵ_1;
- the advantage of *pseudorandom shared key* is bounded by ϵ_2;
- the advantage of *pseudorandom public key* is bounded by ϵ_3.

Note that in the normal lazy sampling of random oracles or permutations, each output will be chosen essentially at random. However, a simulator for indifferentiability will need to occasionally sample in such a way as to be consistent with the ideal NIKE. Next we define two events, named "Strong Consistency Check" (SCC) and "Weak Consistency Check" (WCC). These checks look for the cases where the adversary may already have the necessary information to predict the query response without making the query. If the checks pass, it means the adversary is unable to make such a prediction, and the simulator is essentially free to answer randomly. On the other hand, if the checks fail, the simulator must answer carefully to be consistent with the adversary's prediction.

STRONG CONSISTENCY CHECK. Let $Q_1, \dots, Q_q$ be the sequence of the queries, we say event SCC occurs for the k-th query ($\mathsf{SCC}_k = 1$) if any one of the following cases is satisfied:

- Case 1. The k-th query is P query, say (y, P) and in the previous $k-1$ queries, there is no query with form of $(x, H_0), (x, \mathsf{NKG})$ or $(\mathsf{PK}, x, \mathsf{NSK})$ such that $\mathsf{NKG}(x) = y$.
  Note that if there had been a previous query on such an x, then it would be possible for the adversary to predict $P(y) = P(\mathsf{NKG}(x))$ without making a P query at all by just evaluating $\mathsf{keygen}(H_0(x))$.
- Case 2. The k-th query is P^{-1} query, say (y, P^{-1}) and in the previous $k-1$ queries, there exists no query with form of $(x, H_0), (x, \mathsf{NKG})$ or $(\mathsf{PK}, x, \mathsf{NSK})$ such that $\mathsf{keygen}(\tilde{H}_0(x)) = y$.
  Note that if there had been a previous query on such an x, the adversary can predict $P^{-1}(y) = P^{-1}(\mathsf{keygen}(\tilde{H}_0(x)))$ by querying $\mathsf{NKG}(x)$.

We note, in the ideal world, the simulator is unable to tell if SCC happens since doing so requires knowing the adversary's queries to NKG and NSK. Instead, the simulator will be able to carry out a weak consistency check, WCC:

WEAK CONSISTENCY CHECK. Let $Q_1, \dots, Q_q$ be the sequence of the queries, we say event WCC occurs for the k-th query ($\mathsf{WCC}_k = 1$) if one of the following cases satisfies:

- Case 1. The k-th query is P query, say (y, P) and in the previous $k-1$ queries, there is no query with form of (x, H_0) such that $\mathsf{NKG}(x) = y$.
- Case 2. The k-th query is P^{-1} query, say (y, P^{-1}) and in the previous $k-1$ queries, there is no query with form of (x, H_0) such that $\mathsf{keygen}(\tilde{H}_0(x)) = y$.

- Case 3. The k-th query is H_1 query, say (y_1, y_2, z, H_1), which satisfies $y_1 > y_2$ or in the previous queries, there is no query with form of (x, H_0) such that $\mathsf{NKG}(x) \in \{\mathsf{PK}_1, \mathsf{PK}_2\}$.

This weak consistency check will not catch all the bad cases, but we will demonstrate that the adversary is unable to generate such bad cases, except with negligible probability.

Now we are ready to describe the games. After each game, we give the intuition for why that game is indistinguishable from the previous game. The full proof of indistinguishability between the hybrids appears in Section 7.

Game 0. This game is identical to the real game, where every query is answered by applying real oracles. For instance, the response of (x, H_0) is $\tilde{H}_0(x)$, the response of (x, NKG) is $\tilde{P}^{-1}(\mathsf{keygen}(\tilde{H}_0(x)))$ and so forth. Moreover, during the queries, it also maintains four tables, referring to H_0-table, P-table, P^{-1}-table and H_1-table. Concretely,

- H_0-table: Initially empty, consists of tuples with form of (x, r, y, y'). Once the adversary makes a (x, H_0) query which does not exist in H_0-table (no tuple that the first element of it is x), it inserts $(x, \tilde{H}_0(x), \mathsf{keygen}(\tilde{H}_0(x)), \mathsf{NKG}(x))$ into the table.
- H_1-table: Initially empty, consists of tuples with form of (y_1, y_2, z, w). Once the adversary makes a (y_1, y_2, z, H_1) query that does not exist in H_1-table, it inserts $(y_1, y_2, z, \tilde{H}_1(y_1, y_2, z))$ into the table.
- P-table: Initially empty, consists of tuples with form of $(*, *, y, y')$. Once the adversary makes a (y, P) query which does not exist in P-table, it inserts $(*, *, \tilde{P}(y), y)$ into the table.
- P^{-1}-table: Initially empty, consists of tuples with form of $(*, *, y, y')$. Once the adversary makes a (y, P^{-1}) query which does not exist in P-table, it inserts $(*, *, y, \tilde{P}^{-1}(y))$ into the table.

Note that at this point these tables are just keeping track of information related to adversary's queries, and are completely hidden to the adversary. Next, we define a relation based on these tables. We will say a H_0 query $Q_k = (x, H_0)$ is in a table K, denoted $Q_k \in K$, if there is a 4-tuple in K such that the 1st element is equal to x. Analogously, for a P query we say $Q_k = (y, P) \in K$ if there is a 4-tuple in K such that the 4-th element is equal to y; for P^{-1} queries, we say $Q_k = (y, P^{-1}) \in K$ if there is a 4-tuple in K such that the 3rd element is equal to y. For a H_1-query, we say $Q_k = (y_1, y_2, z, H_1) \in K$ if there is a 4-tuple $T = (T_1, T_2, T_3, T_4)$in K such that $T_1 = y_1, T_2 = y_2, T_3 = z$.

Then, we show a game in which the system responds to the queries by using both tables and the real oracles without changing adversary's view.

Game 1. This game is identical to Game 0, except the way of maintaining the tables and responding the queries. Specifically,

H_0-QUERY. Suppose (x, H_0) is the k-th query($k \in [1, q]$), the system responds as follows:

- Case 1. If $Q_k \in H_0$, then it responds with the corresponding output string;
- Case 2. If $Q_k \notin H_0$ but there is a tuple $T = (T_1, T_2, T_3, T_4)$ in P-table such that $T_4 = \mathsf{NKG}(x)$ and $T_2 \neq *$, then responds with T_2 and inserts (x, T_2, T_3, T_4) into H_0 table;
- Case 3. Otherwise, responds with $\tilde{H}_0(x)$ and inserts $(x, \tilde{H}_0(x), \mathsf{keygen}(\tilde{H}_0(x)),$ $\mathsf{NKG}(x))$ into H_0-table.

P-QUERY. Suppose (y, P) be the k-th query, the system responds as follows:

- Case 1. If $Q_k \in H_0 \cup P \cup P^{-1}$, then responds with the third element of the 4-tuple corresponding to Q_k;

– Case 2. Otherwise, responds with $\tilde{P}(y)$, and inserts $(*, *, \tilde{P}(y), y)$ into both P and P^{-1} table.

P^{-1}-QUERY. Suppose (y, P) be the k-th query, the system responds as follows:

– Case 1. If $Q_k \in H_0 \cup P \cup P^{-1}$, then responds with the fourth element of the 4-tuple corresponding to Q_k;
– Case 2. Otherwise, responds with $\tilde{P}^{-1}(y)$, and inserts $(*, *, y, \tilde{P}^{-1}(y))$ into both P and P^{-1} table.

H_1-QUERY. Suppose (y_1, y_2, z, H_1) be the k-th query, the system responds as follows:

– Case 1: If $Q_k \in H_1$, then it responds with the corresponding output string.
  Note that for the cases below where $Q_k \notin H_1$, after we compute the response w, we will always add the tuple (y_1, y_2, z, w) to H_1-table, so that on future identical queries, Q_k will be in H_1.
– Case 2: If $Q_k \notin H_1$ and $\mathsf{WCC}_k = 1$, then the system would test the validity of z only using P and P^{-1} table. Formally,
  1. If $y_1 > y_2$, then responds with $\tilde{H}_1(y_1, y_2, z)$;
  2. If $y_1 \leq y_2$ and there is a tuple $T = (T_1, T_2, T_3, T_4)$ in P^{-1}-table such that $T_4 = y_1$ and $T_1 \neq *$, then searches every tuple $T' = (T'_1, T'_2, T'_3, T'_4)$ in P-table which satisfies $T'_4 = y_2$ and $T'_2 \neq *$. If the tuple T' is found, then test if $z \overset{?}{=} \mathsf{sharedkey}(T_3, T'_2)$(only using the tuples in $P \cup P^{-1}$). If so, responds with $\mathsf{NSK}(T'_4, T_1)$, otherwise respond with $\tilde{H}_1(y_1, y_2, z)$;
  3. If $y_1 \leq y_2$ and there is a tuple $T = (T_1, T_2, T_3, T_4)$ in P^{-1}-table such that $T_4 = y_2$ and $T_1 \neq *$, then searches every tuple $T' = (T'_1, T'_2, T'_3, T'_4)$ in P-table which satisfies $T'_4 = y_1$ and $T'_2 \neq *$. If the tuple T' is found, then test if $z \overset{?}{=} \mathsf{sharedkey}(T_3, T'_2)$(only using the tuples in $P \cup P^{-1}$). If so, responds with $\mathsf{NSK}(T'_4, T_1)$, otherwise responds with $\tilde{H}_1(y_1, y_2, z)$;
  4. Otherwise, responds with $\tilde{H}_1(y_1, y_2, z)$.
– Case 3: If $Q_k \notin H_1$ and $\mathsf{WCC}_k = 0$, then the system would test the validity of z *only* using H_0-table, P-table and P^{-1}-table. Formally,
  1. Suppose that there is some query $Q_i, i < k$ to H_0 which contains an x such that $\mathsf{NKG}(x_1) = y_1$.
     Suppose there is no such query for y_2. Then searches every tuple in P-table and P^{-1}-table, such that the 4th element of the tuple is y_2. If no such tuple is found, respond to the query with $\tilde{H}_1(y_1, y_2, z)$.
     If a tuple T is found, let T_3 be the 3rd element of T, corresponding to the output of (y_2, P). Then test if $z \overset{?}{=} \mathsf{sharedkey}(T_2, H_0(x_1))$(for the value $H_0(x_1)$, the system uses the tuples in H_0-table). If so, respond with $\mathsf{NSK}(y_2, x_1)$, otherwise respond with $\tilde{H}_1(y_1, y_2, z)$;
  2. The case where there is a query Q_i for y_2 but y_1 is handled analogously.
  3. If both have preimages in H_0, say $\mathsf{NKG}(x_1) = y_1$, $\mathsf{NKG}(x_2) = y_2$, where x_1, x_2 are included in the previous queries, then the system tests $z \overset{?}{=} \mathsf{sharedkey}(\mathsf{keygen}(H_0(x_i)), H_0(x_j))$ (also using the tuples in H_0-table). If valid, then the system responds with $\mathsf{NSK}(y_2, x_1)$, else $\tilde{H}_1(y_1, y_2, z)$.

Note that, in Game 1 the system keeps a longer table, and for each query, it firstly checks whether the tables or honest interfaces can respond it. If can, then it responds with them, else calls the real oracles. In fact, the system in Game 1 uses the tables and the honest interfaces as many as possible and it captures all possible consistency during the queries. Actually, we can treat the system in Game 1 as a *prototype* for our simulator $\mathcal{S}$, and in the following games, we replace the responses that answered by the real oracles with random strings step by step. Moreover, the tuples stored in the table are consistent with the real oracles, hence each response in Game 1 is identical to the one in Game 0.

Game 2. This game is identical to Game 2, except for answering P queries. Assuming (y, P) to be the k-th query, the system responds as follows:

- Case 1: Suppose $Q_k \in H_0 \cup P \cup P^{-1}$, same as above.
- Case 2: If there is no such tuple in $\notin H_0 \cup P \cup P^{-1}$, but $\mathsf{SCC}_k = 1$, then respond with a random string $y' \leftarrow \mathcal{Y}$. The system inserts $(*, *, y', y)$ into the P-table.
- Case 3: If $Q_k \notin H_0 \cup P \cup P^{-1}$ and $\mathsf{SCC}_k = 0$, same as above.

Note that strong consistency check determines whether or not the adversary has the ability to learn $\tilde{P}(y)$ by making a query to NKG. If so, we must answer the P query using $\tilde{P}$; otherwise, we can answer randomly.

Game 3. This game is identical to Game 2, except modifying P-query once more. More concretely, in case 2, if $Q_k \notin P \cup P^{-1}$ and $\mathsf{SCC}_k = 1$, then the system responds with a random public key $y' = \mathsf{keygen}(r)$, where $r \overset{\$}{\leftarrow} \mathcal{X}$ and inserts $(*, *, \mathsf{keygen}(r), y)$ into P-table.

Here, we just change the distribution of outputs in the case where we do not use $\tilde{P}$. Game 3 is indistinguishable from Game 2 by the pseudorandomness of the public key.

Game 4. This game is identical to Game 3, except the way of filling up the P-table. Concretely, in case 2, the system inserts the tuple $(*, r, \mathsf{keygen}(r), y)$ into the table.

Here, the only difference from Game 4 is that we record the random coins r used to sample the output of P into the P table. Since that r is not used with high probability, the adversary's perspective would be close in both games.

Game 5. This game is identical to Game 4, except that to answer P queries, where the system only uses the tables and honest interfaces. Assuming (y, P) to be the k-th query, then

- Case 1. $Q_k \in H_0 \cup P \cup P^{-1}$, same as above.
- Case 2. If $Q_k \notin H_0 \cup P \cup P^{-1}$ and $\mathsf{SCC}_k = 1$, then same as above.
- Case 2. If $Q_k \notin H_0 \cup P \cup P^{-1}$ and $\mathsf{WCC}_k = 1$, then it samples $r \leftarrow \mathcal{X}$, responds with $\mathsf{keygen}(r)$ and inserts $(*, r, \mathsf{keygen}(r), y)$ into P-table.

Game 5 and Game 4 are identical unless the adversary makes a query Q_k such that $Q_k = (y, P), \mathsf{SCC}_k = 0, \mathsf{WCC}_k = 1$ and $Q_k \notin P \cup P^{-1}$. This means that $y = \mathsf{NKG}(x) = \tilde{P}^{-1}(\mathsf{keygen}(\tilde{H}_0(x)))$ for some x, and that the adversary previously queried (x, NKG) or $(\mathsf{PK}, x, \mathsf{NSK})$, but not $(x, \mathsf{H_0})$. Since $Q_k \notin P \cup P^{-1}$, the adversary has never made a query on $(\mathsf{keygen}(\tilde{H}_0(x)), P^{-1})$, as such a query would have resulted in y being added to the table for P^{-1}. Therefore, $\tilde{H}_0(x)$ and thus $\mathsf{keygen}(\tilde{H}_0(x))$ are independent of the adversary's view. In either Game 5 or 4, the query would result in $(*, r, \mathsf{keygen}(\mathsf{r}), y)$ being added to the table for P. The only difference is that in Game 5, $r = \tilde{H}_0$, whereas in Game 5, r is a fresh random value. However, since the adversary has no knowledge of $\tilde{H}_0(x)$, the two games are indistinguishable.

Game 6. This game is identical to Game 5, except the way it answers P^{-1} queries, where the system *only* uses the tables to simulate if the strong consistency check passes. Assuming (y, P^{-1}) to be the k-th query, then,

- Case 1: If $Q_k \in H_0 \cup P \cup P^{-1}$, same as above;
- Case 2: If $Q_k \notin H_0 \cup P \cup P^{-1}$ and $\mathsf{SCC}_k = 1$, then it samples $x \leftarrow \mathcal{X}$, and responds with $\mathsf{NKG}(x)$ and inserts $(x, *, y, \mathsf{NKG}(x))$ into P^{-1}-table.
- Case 3. Otherwise, the system responds with $\tilde{P^{-1}}(y)$.

Note that $\mathsf{SCC}_k = 0$ implies that $Q_k \in H_0$, hence the case $Q_k \notin H_0 \cup P \cup P^{-1}$ and $\mathsf{SCC}_k = 0$ does not exist.

Due to definition, Game 6 and Game 5 are identical unless there exists a query Q_k such that $Q_k = (y, P^{-1})$, $\mathsf{WCC}_k = 1$ and $Q_k \notin P \cup P^{-1}$. This means that, in the previous query, no x appears such that $y = \mathsf{keygen}(\tilde{H}_0(x))$, even in $\mathsf{NKG}, \mathsf{NSK}$ queries, in other words, $\mathsf{NKG}(x)$ never appears and it is hidden from adversary's view. Thus we can replace it with a random value. Moreover, we can replace with $\mathsf{NKG}(\mathsf{x}')$ where $x' \xleftarrow{\$} \mathcal{X}$, as long $\mathsf{keygen}(\tilde{H}_0(x'))$ never appears in the queries. In fact, as x' is randomly sampled and hidden from the adversary, except with negligible probability, $\mathsf{keygen}(\tilde{H}_0(x'))$ never appears.

Game 7. This game is identical to Game 6, except the way it answers P^{-1} queries, where the system *only* uses the tables to simulate, and not the true oracle $\tilde{P}^{-1}$. Assuming (y, P^{-1}) to be the k-th query, then,

- Case 1: If $Q_k \in H_0 \cup P \cup P^{-1}$, same as above;
- Case 2: If $Q_k \notin H_0 \cup P \cup P^{-1}$ and $\mathsf{SCC}_k = 1$, then same as above.
- Case 3: If $Q_k \notin H_0 \cup P \cup P^{-1}$ and $\mathsf{SCC}_k = 0, \mathsf{WCC}_k = 1$, then it samples $x \leftarrow \mathcal{X}$, and responds with $\mathsf{NKG}(x)$ and inserts $(x, *, y, \mathsf{NKG}(x))$ into P^{-1}-table.

Note that $\mathsf{WCC}_k = 0$ implies that $Q_k \in H_0$, hence the case $Q_k \notin H_0 \cup P \cup P^{-1}$ and $\mathsf{WCC}_k = 0$ does not exist.

Due to definition, Game 7 and Game 6 are identical unless there exists a query Q_k such that $\mathsf{SCC}_k = 0, \mathsf{WCC}_k = 1$ and $Q_k \notin P \cup P^{-1}$. This means, there exist queries as (x, NKG) or (y, x, NSK) such that $\mathsf{keygen}(\tilde{H}_0(x)) = y$. Moreover, by Game 7, we also note that query $(\mathsf{NKG}(x), P)$ would leak nothing about $\tilde{H}_0(x)$ and $\mathsf{keygen}(\tilde{H}_0(x))$. Hence, $\tilde{H}_0(x)$ and $\mathsf{keygen}(\tilde{H}_0(x))$ are hidden from the adversary's view. And if such an event happens, it means that the adversary is able to predict the public key for an unknown secret key. As our standard-model signature scheme has psedorandom public keys, this is impossible except with negligible probability.

Game 8. This game is identical to Game 7, except that to answer H_0 queries, where the system only uses the tables and honest interfaces. Assuming (x, H_0) is the k-th query, then

- Case 1. If $Q_k \in H_0$, same as above;
- Case 2. If $Q_k \notin H_0$ but there is a tuple $T = (T_1, T_2, T_3, T_4)$ in P-table such that $T_4 = \mathsf{NKG}(x)$ and $T_2 \neq *$, then same as above;
- Case 3. Otherwise, the system samples $r \leftarrow \mathcal{X}$, responds with r and inserts $(x, r, \mathsf{keygen}(r), \mathsf{NKG}(x))$ into the H_0 table.

Recalling the H_0-table in Game 1, we immediately observe that, the only case that the system calls $\tilde{H}_0(x)$ is when the adversary knows nothing about $\tilde{H}_0(x)$ and $\mathsf{keygen}(\tilde{H}_0)(x)$, although the adversary might know $\mathsf{NKG}(x)$. Therefore, from the adversary's view, $\tilde{H}_0(x)$ is uniformly distributed in $\mathcal{X}$, and it is equivalent to randomly pick $r \leftarrow \mathcal{X}$ and implicitly set $r = (x, H_0)$ and $P^{-1}(\mathsf{keygen}(r)) = \mathsf{NKG}(x)$.

Game 9. This game is identical to Game 8, except when answering the H_1 query. Instead of using a random oracle $\tilde{H}_1$, we now lazily sample the oracle using the table for H_1. However, queries to NSK will still make queries to the random oracle $\tilde{H}_1$. As such, we will need to make certain exceptions where we answer H_1 queries in a way that is consistent with NSK. Specifically, suppose (y_1, y_2, z, H_1) is the k-th query; the system responds as follows:

- Case 1: If $Q_k \in H_1$, then responds the same as above.
- Case 2: If $Q_k \notin H_1$ and $\mathsf{WCC}_k = 1$, then the system would test the validity of z only using P and P^{-1} table. Formally,
  1. If $y_1 > y_2$, then responds with a random string;
  2. If $y_1 \leq y_2$ and there is a tuple $T = (T_1, T_2, T_3, T_4)$ in P^{-1}-table such that $T_4 = y_1$ and $T_1 \neq *$, then searches every tuple $T' = (T_1', T_2', T_3', T_4')$ in P-table which satisfies $T_4' = y_2$ and $T_2' \neq *$. If the tuple T' is found, then test if $z \overset{?}{=} \mathsf{sharedkey}(T_3, T_2')$(only using the tuples in $P \cup P^{-1}$). If so, responds with $\mathsf{NSK}(T_4', T_1)$, otherwise responds with a random string;
  3. If $y_1 \leq y_2$ and there is a tuple $T = (T_1, T_2, T_3, T_4)$ in P^{-1}-table such that $T_4 = y_2$ and $T_1 \neq *$, then searches every tuple $T' = (T_1', T_2', T_3', T_4')$ in P-table which satisfies $T_4' = y_1$ and $T_2' \neq *$. If the tuple T' is found, then test if $z \overset{?}{=} \mathsf{sharedkey}(T_3, T_2')$(only using the tuples in $P \cup P^{-1}$). If so, responds with $\mathsf{NSK}(T_4', T_1)$, otherwise responds with a random string;
  4. Otherwise, responds with a random string.
- Case 3: If $Q_k \notin H_1$ and $\mathsf{WCC}_k = 0$,
  1. Suppose that there is some query $Q_i, i < k$ to H_0 which contains an x such that $\mathsf{KEYGEN}(x) = y_1$.
     Suppose there is no such query for y_2. Then searches the P and P^{-1} tables for the value y_2 in the 4th position, corresponding to an input to P. If no such tuple is found, responds to the query with a random string.
     If a tuple is found, let T be the 3rd element in the tuple, corresponding to the output of P on y_2. Then test if $z \overset{?}{=} \mathsf{sharedkey}(T, \tilde{H}_0(x))$. If so, responds with $\mathsf{SHAREDKEY}(y_2, x)$, otherwise responds with a random string;
  2. The case where there is a query Q_i for y_2 but y_1 is handled analogously.
  3. If both have preimages in the H_0 table, say $\mathsf{KEYGEN}(x_1) = y_1$, $\mathsf{KEYGEN}(x_2) = y_2$, where x_1, x_2 are included in the previous queries, then the system would test $z \overset{?}{=} \mathsf{sharedkey}(\mathsf{keygen}(H_0(x_i)), H_0(x_j))$ (using the tuples in H_0-table). If valid, then the system responds with $\mathsf{SHAREDKEY}(y_2, x_1)$, else a random string.

By definition, we note that in Game 9, the real oracles $\tilde{H}_0, \tilde{P}$ and $\tilde{P^{-1}}$ are hidden, hence expect with negligible probability, the adversary would not output (y_1, y_2, z) such that z is a valid shared key under $\tilde{H}_0, \tilde{P}, \tilde{P^{-1}}$, which means z is a valid shared key for $\mathsf{pk}_1, \mathsf{pk}_2$ where $\mathsf{pk}_\mathsf{i} = \tilde{P}(y_i)$. Now, it's rest to show that, with high probability, for the query (y_1, y_2, z) which fails the consistency check, z is also not a valid shared key for y_1, y_2 under H_0, P, P^{-1}. It's trivial to rule out the cases where the shared-key test fails, and for the rest cases, we observe that, the adversary is obligated to output a valid shared key either without knowing the two secret keys or knowing one secret key but nothing of the other public key. Due to unpredictable shared key and pseudorandom shared key, this is impossible except with negligible probability.

Game 10. In Game 9, the queries to H_0, H_1, P, P^{-1} are answered by the tables which're maintained by the system and by making queries to $\mathsf{NKG}, \mathsf{NSK}$. The system never makes queries directly to $\tilde{H}_0, \tilde{H}_1, \tilde{P}, \tilde{P}^{-1}$; these oracles are *only* used to answer the $\mathsf{NKG}, \mathsf{NSK}$ queries (either generated by the adversary or by the system's response to H_0, H_1, P, P^{-1}

queries). At this point, it is straightforward to show that we can replace $\mathsf{NKG}, \mathsf{NSK}$ with the ideal versions from Definition 4, resulting in Game 10.

We note that in Game 10, the system is efficient, and it responds to the adversarial interfaces just by keeping several tables and accessing the honest interfaces. Thus, we can build a simulator that responds to H_0, H_1, P, P^{-1} queries precisely as the system does in Game 10. The result is that Game 10 corresponds to the ideal world. Since Game 0 is the real world, it suffices to prove that any adjacent games are indistinguishable, and we will give the full description of our simulator and the rest proof in Section 7.

4 Indifferentiable Public Key Encryption

In this section, we propose the notion of "ideal PKE" and then build an indifferentiable public key encryption scheme from ideal NIKE and random oracles. Roughly, our strategy consists of two steps: first, we construct an indifferentiable deterministic public key encryption(DPKE) from an ideal NIKE, and then build an indifferentiable PKE from an ideal DPKE.

4.1 What is Ideal PKE?

In this part, we give the rigorous description of ideal PKE, formally:

Definition 6. (**Ideal PKE.**) *Let $\mathcal{X}, \mathcal{Y}, \mathcal{M}, \mathcal{R}, \mathcal{C}$ be five sets such that:* 1) $|\mathcal{X}|, |\mathcal{Y}|, |\mathcal{R}|, |\mathcal{C}| \geq 2^{\omega(\log\lambda)}$; 2) $|\mathcal{X}| \leq |\mathcal{Y}|$; 3) $|\mathcal{Y}| \times |\mathcal{M}| \times |\mathcal{R}| \leq |\mathcal{C}|$. *We denote $\mathcal{F}[\mathcal{X} \to \mathcal{Y}]$ as the set of all injections that map $\mathcal{X}$ to $\mathcal{Y}$; $\mathcal{E}[\mathcal{Y} \times \mathcal{M} \times \mathcal{R} \to \mathcal{C}]$ as the set of all injections that map $\mathcal{Y} \times \mathcal{M} \times \mathcal{R}$ to $\mathcal{C}$ and $\mathcal{D}[\mathcal{C} \times \mathcal{X} \to \mathcal{M} \cup \perp]$ as the set of all functions that map $\mathcal{X} \times \mathcal{C}$ to $\mathcal{M}$. We define $\mathcal{T}$ as the set of all function tuples (F, E, D) such that:*

- *$F \in \mathcal{F}, E \in \mathcal{E}$ and $D \in \mathcal{D}$;*
- *$\forall x \in \mathcal{X}, m \in \mathcal{M}$ and $r \in \mathcal{R}$, $D(x, E(F(x), m, r)) = m$;*
- *$\forall x \in \mathcal{X}, c \in \mathcal{C}$, if there is no $(m, r) \in \mathcal{M} \times \mathcal{R}$ such that $E(F(x), m, r) = c$, then $D(x, c) = \perp$.*

We say that a PKE scheme $\Pi_{\mathsf{PKE}} = (\mathsf{PKE.KEYGEN}, \mathsf{PKE.ENC}, \mathsf{PKE.DEC})$, associated with secret key space $\mathcal{X}$, public space $\mathcal{Y}$, message space $\mathcal{M}$, nonce space space $\mathcal{R}$, and ciphertext space $\mathcal{C}$, is an ideal PKE if Π_{PKE} is sampled from $\mathcal{T}$ uniformly. Moreover, if the nonce space is empty and the message space is sufficiently large, then we say such a scheme is an ideal DPKE.

Same as in Section 3, we note that, due to the information-theoretic argument, an ideal PKE achieves CCA-2 secure, related-key attack secure, leakage-resilient and so forth. Next, we show how we construct an indifferentiable PKE scheme, from an ideal NIKE.

4.2 Construction for Deterministic PKE

In this section, we build an indifferentiable *deterministic* PKE (DPKE) from simpler ideal primitives (namely random oracles and ideal ciphers) along with an ideal NIKE. We firstly present our first attempt of the construction and then illustrate a differentiator to break it (this attack also indicates a difficulty of building indifferentiable PKE). Next, we give our solution to get rid of the attack and establish the proof.

First attempt to build an indifferentiable DPKE. Given an ideal NIKE Π_{NIKE}, a natural way to build an indifferentiable DPKE is the follwing. Firstly, convert this ideal NIKE into a PKE scheme, and then apply the Fujisaki-Okamoto transformation [20], which combines with a random oracle to give at least CCA-2 security. The hope is that this transformation would give us an indiffernetiable DPKE. Specifically, let $\Pi_{\mathsf{NIKE}} = (\mathsf{NKG}, \mathsf{NSK})$ be an ideal NIKE, associated with secret key space $\mathcal{X}$, public key space $\mathcal{Y}$ and shared key space $\mathcal{Z}$. Let $\mathsf{H}_0, \mathsf{H}_1 := \{0,1\}^* \to \mathcal{X}; \mathsf{H}_2 := \{0,1\}^* \to \mathcal{Z}; \mathsf{H}_3 := \{0,1\}^* \to \mathcal{M}$, then applying FO-transform, we have the following DPKE scheme: $\Pi_{\mathsf{DPKE}} = (\mathsf{DPKE.KEYGEN}, \mathsf{DPKE.ENC}, \mathsf{DPKE.DEC})$.

- $\mathsf{DPKE.KEYGEN}(\mathsf{SK})$: On inputs secret key SK, the algorithm outputs public key $\mathsf{PK} = \mathsf{NKG}(H_0(\mathsf{SK}))$;
- $\mathsf{DPKE.ENC}(\mathsf{PK}, \mathsf{M})$: On inputs public key PK and message M, the algorithm computes $\delta = H_2(\mathsf{PK}||\mathsf{M})$, and outputs ciphertext C as

$$\mathsf{C} = (\mathsf{C}_1, \mathsf{C}_2, \mathsf{C}_3) = (\mathsf{NKG}(H_1(\mathsf{PK}||\mathsf{M})), \delta \oplus \mathsf{NSK}(\mathsf{PK}, H_1(\mathsf{PK}||\mathsf{M})), H_3(\mathsf{PK}, \delta) \oplus \mathsf{M});$$

- $\mathsf{DPKE.DEC}(\mathsf{SK}, \mathsf{C})$: On inputs secret key SK and ciphertext $\mathsf{C} = (\mathsf{C}_1, \mathsf{C}_2, \mathsf{C}_3)$, the algorithm computes:

$$\mathsf{PK} = \mathsf{DPKE.KEYGEN}(\mathsf{SK}); \mathsf{A}_1 = \mathsf{NSK}(\mathsf{C}_1, H_0(\mathsf{SK}));$$
$$\mathsf{A}_2 = \mathsf{C}_2 \oplus \mathsf{A}_1; \mathsf{A}_3 = \mathsf{C}_3 \oplus H_3(\mathsf{PK}||\mathsf{A}_2).$$

Then it tests whether $\mathsf{C} \overset{?}{=} \mathsf{DPKE.ENC}(\mathsf{PK}||\mathsf{M})$. If yes, then outputs A_3, else aborts.

Correctness easily follows, but this scheme is not indifferentiable. Next we present a differetiator which breaks the scheme.

Differentiator for FO transform. Here we give the description of the differentiator $\mathcal{D}$ that breaks the security of the scheme above. Due to definition, $\mathcal{D}$ has three honest interfaces $(\mathsf{DPKE.KEYGEN}, \mathsf{DPKE.ENC}, \mathsf{DPKE.DEC})$(below, we will denote $(\mathsf{DKG}, \mathsf{DE}, \mathsf{DD})$ for short) and six adversarial interfaces $(H_0, H_1, H_2, H_3, \mathsf{NKG}, \mathsf{NSK})$, and $\mathcal{D}$ works:

```
Differentiator $\mathcal{D}$:
$\mathsf{sk} \xleftarrow{\$} \mathcal{X}, \mathsf{M} \xleftarrow{\$} \mathcal{M};$
$\mathsf{A} \leftarrow \mathsf{DKG}(\mathsf{sk}), (\mathsf{B}_1, \mathsf{B}_2, \mathsf{B}_3) \leftarrow \mathsf{DE}(\mathsf{A}, \mathsf{M});$
$\mathsf{Q}_1 \leftarrow \mathsf{H}_0(\mathsf{sk}), \mathsf{Q}_2 \leftarrow \mathsf{NSK}(\mathsf{B}_1, \mathsf{Q}_1), \mathsf{Q}_3 = \mathsf{H}_3(\mathsf{A}, \mathsf{Q}_2 \oplus \mathsf{B}_2);$
Return $1(\mathsf{Q}_3 = \mathsf{B}_3 \oplus \mathsf{M})$
```

Fig. 2: Differentiaor for FO-transform.

We immediately observe that, in the real game, A and $(\mathsf{B}_1, \mathsf{B}_2, \mathsf{B}_3)$ are the corresponding public key and ciphertext, respectively. Moreover, as Π_{NIKE} is an ideal NIKE, we have

$$\mathsf{NSK}(\mathsf{PK}, \mathsf{H}_1(\mathsf{PK}||\mathsf{M})) = \mathsf{NSK}(\mathsf{C}_1, \mathsf{H}_0(\mathsf{sk})) = \mathsf{NSK}(\mathsf{B}_1, \mathsf{Q}_1).$$

which refers to,

$$\Pr[\mathsf{Q}_3 = \mathsf{H}_3(\mathsf{PK}, \delta) = \mathsf{B}_3 \oplus \mathsf{M}] = 1.$$

However, in the ideal game, the simulator would have no information of the honest queries that $\mathcal{D}$ calls. Based on the description above, we note that the simulator only has the information as follows: $(\mathsf{sk}, \mathsf{A}, \mathsf{B}_1, \mathsf{B}_2)$(other information such like $\mathsf{H}_0(\mathsf{sk}), \mathsf{NSK}(\mathsf{B}_1, \mathsf{Q}_1)$ and $\mathsf{H}_3(\mathsf{A}, \mathsf{Q}_2 \oplus \mathsf{B}_2)$ are simulated by $\mathcal{S}$ itself). Therefore, without decryption oracle, M is independent of simulator's view. And the decryption oracle always aborts except $\mathcal{S}$ hands in a valid ciphertext, which consists of three elements $(\mathsf{B}_1, \mathsf{B}_2, \mathsf{B}_3)$. Moreover, in the ideal world, the honest interface $\mathsf{DPKE.ENC}$ is a random injection, hence

$$\Pr[\mathcal{S} \text{ outputs a valid } \mathsf{B}_3] \leq \frac{\mathsf{poly}(\lambda)}{|\mathcal{M}|},$$

which means, in the ideal game,

$$\Pr[\mathsf{Q}_3 = \mathsf{H}_3(\mathsf{A}, \mathsf{Q}_2 \oplus \mathsf{B}_2) = \mathsf{B}_3 \oplus \mathsf{M}] \leq \frac{\mathsf{poly}(\lambda)}{|\mathcal{M}|}.$$

Difficulty of building indifferentiable PKE. Due to this differentiator, we know that: if we want to build an indifferentiable PKE, we should be super-cautious if the ciphertext includes piece with form of $\mathsf{string} \oplus \mathsf{M}$ [6]. And interestingly, most of the known PKE schemes have this kind of piece in ciphertext. Next, we give our solution to get rid of this kind of attack.

Our solution. To prevent the attack above, the only hope is $\mathcal{S}$ can always output a valid ciphertext itself. Our trick is, instead of using random oracles, we use an ideal cipher model P, with inverse. Specifically, let $\mathcal{Z} = \mathcal{Y} \times \mathcal{M}$ (we specify the shared key space of Π_{NIKE} to be $\mathcal{Y} \times \mathcal{M}$ and $|\mathcal{X}| \leq |\mathcal{M}|$), and $P := \mathcal{Z} \to \mathcal{Z}$. Now, we denote $\delta = P(\mathsf{PK}||\mathsf{M})$, and modify the ciphertext as $C = (\mathsf{NKG}(\mathsf{H}_1(\mathsf{PK}||\mathsf{M})), \delta \oplus \mathsf{NSK}(\mathsf{PK}, \mathsf{H}_1(\mathsf{PK}||\mathsf{M}))))$. Formally:

- $\mathsf{DPKE.KEYGEN}(\mathsf{SK})$: On inputs secret key SK, the algorithm outputs public key $\mathsf{PK} = \mathsf{NKG}(H_0(\mathsf{SK}))$;
- $\mathsf{DPKE.ENC}(\mathsf{PK}, \mathsf{M})$: On inputs public key PK and message M, the algorithm computes $\delta = P(\mathsf{PK}||\mathsf{M})$, and outputs ciphertext C as
$$\mathsf{C} = (\mathsf{C}_1, \mathsf{C}_2) = (\mathsf{NKG}(H_1(\mathsf{PK}||\mathsf{M})), \delta \oplus \mathsf{NSK}(\mathsf{PK}, \mathsf{H}_1(\mathsf{PK}, \mathsf{M})));$$
- $\mathsf{DPKE.DEC}(\mathsf{SK}, \mathsf{C})$: On inputs secret key SK and ciphertext $\mathsf{C} = (\mathsf{C}_1, \mathsf{C}_2)$, the algorithm computes:
$$\mathsf{PK} = \mathsf{DPKE.KEYGEN}(\mathsf{SK}); \mathsf{A}_1 = \mathsf{NSK}(\mathsf{C}_1, H_0(\mathsf{SK}));$$
$$\mathsf{A}_2 = \mathsf{C}_2 \oplus \mathsf{A}_1; \mathsf{A}_3 = P^{-1}(A_2), \mathsf{A}_4 = \mathsf{A}_3/\mathsf{PK}.$$

Then it tests whether $\mathsf{C} \overset{?}{=} \mathsf{DPKE.ENC}(\mathsf{A}_3)$. If yes, then outputs A_4[7], else aborts.

In our new setting, we immediately observe that the ciphertext only consists of *two* elements, so $\mathcal{S}$ can hand in the vaild ciphertext to the decryption oracle. Intuitively, following the same spirit, the corresponding differentiator for the new scheme should be:

Same as above, in real game, $\Pr[\mathcal{D}] = 1$. Meanwhile, in ideal game, we note that $\mathcal{S}$ still can extract the following information $(\mathsf{sk}, \mathsf{A}, \mathsf{B}_1, \mathsf{B}_2)$. As now the ciphertext only consists of two elements, $\mathcal{S}$ can run the decryption oracle $\mathsf{M}' \leftarrow \mathsf{DD}(\mathsf{sk}, (\mathsf{B}_1, \mathsf{B}_2))$ and output $\mathsf{Q}_3 = \mathsf{A}||\mathsf{M}'$. Hence, in ideal game, we also have $\Pr[\mathcal{D}] = 1$. Apparently, this is *only* an evidence that our new scheme prevents this specific differentiator. And to prove it's an indifferentiable DPKE, we need to show that our scheme can prevent all PPT differetiators. Next, we prove the following theorem.

Theorem 7. (Indifferentiable DPKE). *Π_{DPKE} is indifferentiable from an ideal DPKE if Π_{NIKE} is an ideal NIKE.*

As the proof of Theorem 7 is long, we will give the full details in Section 8.

[6] we stress that this difficulty might not be an impossibility separation, and here we just point out a vulnerability of building indifferentiable PKEs

[7] we note that $\mathsf{A}_3 = \mathsf{PK}||\mathsf{M}$, and by A_3/PK, we mean removing PK from A_3.

```
Differentiator $\mathcal{D}$:
$\mathsf{sk} \xleftarrow{\$} \mathcal{X}, \mathsf{M} \xleftarrow{\$} \mathcal{M}$;
$\mathsf{A} \leftarrow \mathsf{DKG}(\mathsf{sk}), (\mathsf{B}_1, \mathsf{B}_2) \leftarrow \mathsf{DE}(\mathsf{A}, \mathsf{M})$;
$\mathsf{Q}_1 \leftarrow \mathsf{H}_0(\mathsf{sk}), \mathsf{Q}_2 \leftarrow \mathsf{NSK}(\mathsf{B}_1, \mathsf{Q}_1), \mathsf{Q}_3 = \mathsf{P}^{-1}(\mathsf{Q}_2 \oplus \mathsf{B}_2)$;
Return $1(\mathsf{Q}_3 = \mathsf{A}||\mathsf{M})$
```

Fig. 3: Differentiaor for new scheme.

4.3 Construction for PKE

In this section, we complete the construction by building an indifferentiable PKE from ideal DPKE and random oracles. Similarly as in Section 4.2 , we firstly present two attempts and then illustrate the corresponding differentiators to break the schemes. Then we give the modified solution to get rid of the attacks and complete the proof.

First attempt. Given an ideal DPKE $\Pi_{\mathsf{DPKE}} = (\mathsf{DKG}, \mathsf{DE}, \mathsf{DD})$, a trivial way to build a PKE scheme is the following:

- $\mathsf{PKE.KEYGEN(SK)} = \mathsf{DKG(SK)}$;
- $\mathsf{PKE.ENC(PK, M, R)} = \mathsf{DE(PK, M||R)}$;
- $\mathsf{PKE.DEC(SK, C)}$: On inputs a secret key SK and a ciphertext C, the algorithm runs $\mathsf{DD(SK, C)}$. If aborts then it aborts, else let $(\mathsf{M}||\mathsf{R}) = \mathsf{DD(SK, C)}$, it outputs M.

Differentiator for first attempt. Here we build a differentiator $\mathcal{D}$ to break the scheme above. Due to definition, $\mathcal{D}$ has three honest interfaces ($\mathsf{PKE.KEYGEN}$, $\mathsf{PKE.ENC}, \mathsf{PKE.DEC}$)(below, we will denote ($\mathsf{PKG}, \mathsf{PE}, \mathsf{PD}$) for ease) and three adversarial interfaces ($\mathsf{DKG}, \mathsf{DE}, \mathsf{DD}$). We build $\mathcal{D}$ as follows:

```
Differentiator $\mathcal{D}$:
$\mathsf{sk} \xleftarrow{\$} \mathcal{X}, \mathsf{M} \xleftarrow{\$} \mathcal{M}; \mathsf{R} \xleftarrow{\$} \mathcal{R}$;
$\mathsf{A} \leftarrow \mathsf{PKG}(\mathsf{sk}), \mathsf{B} \leftarrow \mathsf{PE}(\mathsf{A}, \mathsf{M}, \mathsf{R}),\ \mathsf{Q}_1 \leftarrow \mathsf{DD}(\mathsf{sk}, \mathsf{B})$
Return $1(\mathsf{Q}_1 = \mathsf{M}||\mathsf{R})$
```

Fig. 4: Differentiaor for first attempt.

Easy to note that, in real game $\Pr[\mathcal{D} = 1] = 1$. Meanwhile, in ideal game, the simulator has no information of the randomness R, as PD only outputs M, which means $\Pr[\mathcal{D} = 1] \leq \frac{1}{|\mathcal{R}|}$.

Second attempt. To solve the problem above, we add a random oracle into our construction. Specifically, let $H_0 := \{0,1\}^* \to \mathcal{R}$, then we build Π_{PKE} as:

- $\mathsf{PKE.KEYGEN(SK)} = \mathsf{DKG(SK)}$;
- $\mathsf{PKE.ENC(PK, M, R)} = \mathsf{DE}(\mathsf{PK}, \mathsf{M}||H_0(\mathsf{PK}, \mathsf{M}, \mathsf{R}))$;
- $\mathsf{PKE.DEC(SK, C)}$: On inputs a secret key SK and a ciphertext C, the algorithm runs $\mathsf{DD(SK, C)}$. If aborts then it aborts, else let $(\mathsf{M}||\mathsf{str}) = \mathsf{DD(C, SK)}$, it outputs M.

Differentiator for second attempt. Although we add a random oracle H_0, this attempt still fails. Here is the differentiator $\mathcal{D}$:

Easy to see that, in real game, $\Pr[\mathcal{D} = 1] = 1$. However, in ideal game, with noticeable probability, the decryption would abort. In fact, iff there exists a nonce $R \in \mathcal{R}$ such that $H_0(\mathsf{PK}, \mathsf{M}, \mathsf{R}) = \mathsf{r}$, the decryption outputs M. Moreover, as $\mathsf{R}, \mathsf{r} \in \mathcal{R}$, and H_0 is a random oracle, we have that

$$\Pr[\forall \mathsf{R} \in \mathcal{R}, H_0(\mathsf{PK}, \mathsf{M}, \mathsf{R}) \neq r] \approx 1/e$$

```
Differentiator $\mathcal{D}$:
$\mathsf{sk} \xleftarrow{\$} \mathcal{X}, \mathsf{M} \xleftarrow{\$} \mathcal{M}; \mathsf{r} \xleftarrow{\$} \mathcal{R}$;
$\mathsf{A} \leftarrow \mathsf{PKG}(\mathsf{sk}), \mathsf{B} \leftarrow \mathsf{DE}(\mathsf{A}, \mathsf{M}||\mathsf{r})$, $\mathsf{Q}_1 \leftarrow \mathsf{PD}(\mathsf{sk}, \mathsf{B})$
Return $1(\mathsf{Q}_1 \neq \perp)$
```

Fig. 5: Differentiaor for second attempt.

which refers to $\Pr[\mathcal{D} = 1] \leq 1 - 1/e \approx 0.6$

Our solution. To prevent the attack above, we have to shorten the size of H_0's range, to make sure that every element in H_0's range has pre-image with high probability. Meanwhlie, to make sure the ciphertext space is sufficiently large, we also need to pad some dummy strings. Specifically, let Π_{DPKE} be an ideal DPKE, associated with secret key space $\mathcal{X}$, public key space $\mathcal{Y} = \{0,1\}^{n_1}$, message space $\mathcal{M} = \{0,1\}^{n_2+2n_3}$, and ciphertext space $\mathcal{C}$, and let $H_0 := \{0,1\}^* \to \{0,1\}^{n_3}$, where $n_2 > 0$ and $n_1, n_3 \geq \omega(\log \lambda)$. Then we build our scheme as:

- $\mathsf{PKE.KEYGEN}(\mathsf{SK}) = \mathsf{DKG}(\mathsf{SK})$:
- $\mathsf{PKE.ENC}(\mathsf{PK}, \mathsf{M}, \mathsf{R})$: On inputs public key PK, message $\mathsf{M} \in \{0,1\}^{n_2}$ and nonce $\mathsf{R} \in \{0,1\}^{2n_3}$, the algorithm outputs ciphertext:

$$\mathsf{C} = \mathsf{DE}(\mathsf{PK}, \mathsf{M}||H_0(\mathsf{PK}, \mathsf{M}, \mathsf{R})||\underbrace{0\ldots0}_{n_3});$$

- $\mathsf{PKE.DEC}(\mathsf{SK}, \mathsf{C})$: On inputs secret key SK and ciphertext C, the algorithm runs $\mathsf{A} = \mathsf{D}_{\mathsf{DPKE}}(\mathsf{SK}, \mathsf{C})$. If $\mathsf{A} = \perp$ or the last n_3 bits are not 0^{n_3}, then it aborts, else the algorithm outputs the first n_2 bits.

Correctness follows easily from ideal DPKE's definition and the rest is to prove our solution is indifferentiable.

Theorem 8. (Indifferentiable PKE). Π_{PKE} *is indifferentiable from an ideal PKE if* Π_{DPKE} *is an ideal DPKE.*

Again, as the proof of Theorem 8 is quite long, we will give the full details in Section 9.

5 Indifferentiable Digital Signatures

In this section, we extend our result to the digital signature scheme. We propose the notion of "Ideal Signature", and then build an indifferentiable signature scheme from simpler ideal primitives and a stand-model signature scheme.

5.1 What is "Ideal Signature"?

In this part, we give the rigorous description of ideal signature, formally:

Definition 9. (**Ideal Signature.**) *Let $\mathcal{X}, \mathcal{Y}, \mathcal{M}, \Sigma$ be four sets such that:* 1) $|\mathcal{X}|, |\mathcal{Y}|, |\mathcal{M}|, |\Sigma| \geq 2^{\omega(\log \lambda)}$*;* 2) $|\mathcal{X}| \leq |\mathcal{Y}|$*;* 3) $|\mathcal{X}| \times |\mathcal{M}| \leq |\Sigma|$*. We denote $\mathcal{F}[\mathcal{X} \to \mathcal{Y}]$ as the set of all injections that map $\mathcal{X}$ to $\mathcal{Y}$; $\mathcal{S}[\mathcal{X} \times \mathcal{M} \to \Sigma]$ as the set of all injections that map $\mathcal{X} \times \mathcal{M}$ to Σ and $\mathcal{V}[\mathcal{Y} \times \mathcal{M} \times \Sigma \to \{0,1\}]$ as the set of all functions that map $\mathcal{Y} \times \mathcal{M} \times \Sigma$ to a bit. We define $\mathcal{T}$ as the set of all function tuples (F, S, V) such that:*

- *$F \in \mathcal{F}, S \in \mathcal{S}$ and $V \in \mathcal{V}$;*
- *$\forall x \in \mathcal{X}, m \in \mathcal{M}$, $V(F(x), m, S(x, m)) = 1$;*

- $\forall x \in \mathcal{X}, m \in \mathcal{M}$ *and* $\sigma \in \Sigma$, *if* $\sigma \neq \mathsf{sign}(x, m)$, *then* $V(F(x), m, \sigma) = 0$*;*
- $\forall x \in \mathcal{X}, m \in \mathcal{M}$ *and* $\sigma_1, \sigma_2 \in \Sigma$, $V(F(x), m, \sigma_1) = V(F(x), m, \sigma_2) = 1 \Rightarrow \sigma_1 = \sigma_2$.

We say that a digital signature scheme $\Pi_{\mathsf{DS}} = (\mathsf{DS.KEYGEN}, \mathsf{DS.SIGN}, \mathsf{DS.VER})$*, associated with secret key space* $\mathcal{X}$*, public key space* $\mathcal{Y}$*, message space* $\mathcal{M}$ *and signature space* Σ*, is an ideal digital signature, if* Π_{DS} *is sampled from* $\mathcal{T}$ *uniformly.*

It's trivial to note that, an ideal signature is a unique signature, and due to the information-theoretic argument, it also achieves CMU-secure, related-key attack secure, leakage-resilient and so forth. Next, we show how to construct an indifferentiable signature scheme from simpler ideal primitives.

5.2 Construction

In this section, we build our indifferentiable signature scheme from simpler ideal primitives (namely random oracles and ideal ciphers) along with a standard-model(that is, *non-ideal*) signature scheme.

Building Blocks. Our scheme will consist of several building blocks:

- A standard-model signature scheme $\Pi_{\mathsf{SM-Sig}} = (\mathsf{keygen}, \mathsf{sign}, \mathsf{ver})$ with secret key space $\mathcal{X}$, public key space $\mathcal{Y} = \{0,1\}^{n_1}$, message space $\mathcal{M} = \{0,1\}^{n_2}$ and signature space $\Sigma \subset \{0,1\}^{n_3}$;
- $H_0 := \{0,1\}^* \to \mathcal{X}; H_1 := \{0,1\}^* \to \mathcal{M}$;
- $P := \{0,1\}^{n_1} \to \{0,1\}^{n_1}$ is a random permutation and P^{-1} is P's inverse,
- $E := \{0,1\}^{n_1+n_2} \times \{0,1\}^{n_3} \to \{0,1\}^{n_3}$ is an ideal cipher model, where $\{0,1\}^{n_1+n_2}$ is its key space and E^{-1} is its inverse.

Construction. Now we are ready to build an indifferentiable signature scheme, denoted as $\Pi_{\mathsf{Sig}} = (\mathsf{Sig.KEYGEN}, \mathsf{Sig.SIGN}, \mathsf{Sig.VER})$, from the building blocks above. Formally,

- $\mathsf{Sig.KEYGEN(SK)}$: On inputs secret key SK, the algorithm outputs public key $\mathsf{PK} = P^{-1}(\mathsf{keygen}(H_0(\mathsf{SK})))$;
- $\mathsf{Sig.SIGN(SK, M)}$: On inputs secret key SK and message M, the algorithm computes $\mathsf{PK} = \mathsf{Sig.KEYGEN(SK)}$ and outputs the signature
$$\sigma = E^{-1}((\mathsf{PK}||\mathsf{M}), \mathsf{sign}(H_0(\mathsf{SK}), H_1(\mathsf{M}))),$$
- $\mathsf{Sig.VER}(\mathsf{PK}, \mathsf{M}, \sigma)$: On inputs public key PK, message M and the signature σ, the algorithm outputs a bit $b = \mathsf{ver}(P(\mathsf{PK}), H_1(\mathsf{M}), E((\mathsf{PK}||\mathsf{M}), \sigma))$.

Correctness of the scheme easily follows, and what's more interesting is its indifferentiability. Next, we prove our scheme is indifferentiable from an ideal signature. Before that, we firstly specify the security properties of the standard-model signature.

Property 1. UNIQUENESS. We say a signature achieves uniqueness, if $\forall(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{keygen}, \mathsf{m} \in \mathcal{M}, \sigma_1, \sigma_2 \in \Sigma$, we have,

$$\mathsf{ver}(\mathsf{pk}, \mathsf{m}, \sigma_1) = \mathsf{ver}(\mathsf{pk}, \mathsf{m}, \sigma_2) = 1 \Rightarrow \sigma_1 = \sigma_2.$$

Property 2. RANDOM-MESSAGE ATTACK (RMA). We say a signature scheme is RMA-secure if for any PPT adversary $\mathcal{A}$, the advantage

$$\mathrm{Adv}_{\mathcal{A}} := \Pr[\mathsf{ver}(\mathsf{pk}, \mathsf{m}^*, \sigma^*) = 1 : \sigma^* \leftarrow \mathcal{A}^{\mathsf{sign}(\mathsf{sk}, \mathsf{m}_1, \ldots, \mathsf{m}_\mathsf{q})}(\mathsf{pk}, \mathsf{m}^*)] \leq \mathsf{negl}(\lambda)$$

where $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{keygen}$, $(\mathsf{m}^*, \mathsf{m}_1, \ldots, \mathsf{m}_\mathsf{q}) \xleftarrow{\$} \mathcal{M}$ and m^* was not previously signed.

Property 3. PSEUDORANDOM PUBLIC KEY. We say the public key, for a signature scheme, is pseudorandom, if for any PPT adversary $\mathcal{A}$, the advantage

$$\mathrm{Adv}_{\mathcal{A}} := |\Pr[\mathcal{A}(\mathsf{keygen}(r))] - \Pr[\mathcal{A}(R)]| \leq \mathsf{negl}(\lambda)$$

where $r \leftarrow \mathcal{X}, R \leftarrow \mathcal{Y}$.

We say a signature scheme is Good if it satisfies the three properties above. Next, we prove the following theorem.

Theorem 10. (Indifferentiable Signatures). Π_{Sig} *is indifferentiable from an ideal digital signature if* $\Pi_{\mathsf{SM-Sig}}$ *is* Good.

Again, as the proof of Theorem 10 is quite long, we will give the full details in Section 10.

6 Instantiating Our Constructions

In this section, we explain how to instantiate our schemes. In particular, we briefly explain how to build secure standard-model schemes that can be plugged into our construction.

6.1 Non-interactive Key Exchange (NIKE)

We need a NIKE protocol where (1) the shared key is computationally unpredictable; (2) for any secret key, the distribution of shared key is close to uniform in shared key space over the public key's distribution and (3) the public values exchanged by the parties are computationally close to uniform bit strings. (1) is a weakening of the usual notion of security for a NIKE scheme, and (2) is guaranteed in most known schemes, except for some artificial constructions. In contrast, (2) is non-standard, and requires some care to make sure it is handled properly. We now give several example constructions:

From CDH. The usual Diffie-Hellman protocol gives a NIKE that is almost good enough for us. Let G be a group of prime order p with generator g. The secret keys are integers $a \in \mathbb{Z}_p$. $\mathsf{keygen}(a) = g^a$ and $\mathsf{sharedkey}(h, a) = h^a$. For $g^a = \mathsf{keygen}(a)$ and $g^b = \mathsf{keygen}(b)$, the secret shared key is $\mathsf{sharedkey}(g^a, b) = g^{ab}$. The computational Diffie-Hellman assumption states that is is computationally infeasible to compute g^{ab} from g^a, g^b. This immediately shows the unpredictability of the shared key in the NIKE protocol.

For the pseudorandomness of public keys, we need some more care. We consider two cases:

– G is the group of quadratic residues mod a safe prime q. In other words, $q = 2p + 1$. In this case, the public key is a random residue, clearly not a uniformly random bit string
– G is an elliptic curve over a finite field. Here, the public key is a random point on the elliptic curve, which is represented by two elements of the finite field satisfying the elliptic curve equation. Again, clearly not a uniform bit string.

We now explain how to convert either protocol into one with statistically random public keys.

In the case of the group of quadratic residues, we will assume $q = 3 \bmod 4$. In this case, -1 is *not* a residue, meaning that for every non-zero integer x in $\mathbb{Z}_q$, exactly one of x and $-x$ is a reside. As such, we can bijectively map the group G to the integers $\{1, \ldots, p\}$: for each element $g \in G$, if $g \in \{1, \ldots, p\}$ output g, and otherwise output $q - g$.

We are still not done: now the public key is random in $\{1, \ldots, p\}$, but as p is not a power of 2, this cannot be represented as a uniform bit string. Instead, we will use the following:

Lemma 11. *Let D be a distribution over $[0, N-1]$ that is computationally indistinguishable from uniform. Then n be an integer such that $n - \log_2 N \geq \lambda$ where λ is the security parameter. Let T be the largest integer such that $TN \leq 2^n$. Then the distribution $x + Ny$ where $x \leftarrow S$ and $y \leftarrow [0, T-1]$ is computationally indistinguishable from the uniform distribution over $[0, 2^n - 1]$.*

Proof. We prove this by a sequence of hybrids:

Hybrid 0. In this case, the adversary is given $z = x + Ny$ for $x \leftarrow D$ and $y \leftarrow [0, T-1]$.

Hybrid 1. In this case, the adversary is given $z = x + Ny$ for $x \leftarrow [0, N-1], y \leftarrow [0, T-1]$. Notice that Hybrids 0 and 1 are computationally indistinguishable since D is computationally indisitnguishable from uniform on $[0, N-1]$. Notice that in Hybrid 2, z is uniform on $[0, TN-1]$

Hybrid 2. In this hybrid, z is uniform on $[0, 2^n]$. Notice that $\|2^n - TN\| \leq N$. Therefore the two distributions are at most $N/2^n$-close in statistical distance. Notice that $N/2^n \leq \frac{1}{2^\lambda}$, which is negligible.

Lemma 11 shows that any key generation algorithm where outputs are uniform in some interval can be turned into an algorithm where outputs are uniform bit strings. By applying this Lemma to Diffie-Hellman over safe primes, we obtain a NIKE scheme meeting our needs.

In the case of elliptic curves, things are a bit more complicated. Fortunately, there are multiple ways to represent elliptic curve elements as (almost) random elements in the ambient finite field; for example, see [30]. Elements in the finite field of q elements can be mapped into the integers $[0, q-1]$ in a natural way, and then we can apply Lemma 11.

From LWE. We can also construct our NIKE scheme from LWE, using the standard LWE-based NIKE, but with some modifications.

The standard NIKE scheme works as follows:

- First, a setup phase picks a random wide matrix $A \in \mathbb{Z}_q^{n \times m}$ where $m \gg n$.
- Next, Alice chooses a random value $x \in \{0,1\}^m$, and computes $u = A \cdot x$
- Then, Bob chooses a random $s \in \mathbb{Z}_q^n$ and a random vector $e \in \mathbb{Z}_q^m$ such that the entries of e come from discrete Gaussians of width much smaller than q. Bob outputs $v = s \cdot A + e$
- The shared key consists of the most significant bits of $s \cdot A \cdot x$. Bob computes this as $s \cdot u$. Meanwhile, Alice can compute $v \cdot x = s \cdot A \cdot x + s \cdot x$. Since $s \cdot x$ is small, the most significant bits of this value match $s \cdot A \cdot x$.

The pseudorandomness (and hence computational unpredictability) of the shared key follows from the LWE assumption, which states that the pair (A, v) is computationally indisitinguishable from truly random values over $\mathbb{Z}_q^{(n+1) \times m}$.

There are several problems with us applying this scheme:

- There is a setup phase to produce A. But this is not really a concern for us: we simply assume one more random oracle J, and let $A = J(0)$. In other words, as long as the setup outputs a random value, such a setup is "free" in the random oracle model.
- Alice and Bob apply different algorithms. This is easily handled by simply having Alice and Bob run two independent instances of the protocol, where they play opposite roles in each instance. The final key is just the XOR of the keys from each instance.
- By the Leftover Hash Lemma, for an appropriate choice of n, m, Alice's message u is statistically close to random in $\mathbb{Z}_q^n$, even give A. By LWE, Bob's message v is computationally indisitinguishable from random in $\mathbb{Z}_q^m$ given A. We need to map these to uniformly random bit strings. First, we map u into $[0, q^n - 1]$ and v into $[0, q^m - 1]$. Then we apply Lemma 11 to sample the actual public keys as statistically-close to uniform bit strings.

By making all the necessary modifications, the LWE-based NIKE satisfies all the needed criteria to be used in our scheme.

6.2 Signatures

Next, we turn to signatures. Unfortunately, as we need unique signatures, we have to be careful about how we instantiate our scheme.

From Bilinear maps. We can use the unique signature scheme of Boneh, Lynn, and Shacham [7] from bilinear maps. Since we need a weaker notion of security than the usual chosen message attack security, we can actually use a simplification of their scheme.

The secret key is an integer $a \in \mathbb{Z}_p$, and the public key is $h = g^a$. Messages m are just group elements, and the signature on m is $\sigma = m^a$. Notice that (g, h, m, σ) is a Diffie-Hellam tuple; therefore, all that's needed to verify the signature is a way to solve the decisional Diffie-Hellam (DDH) problem. On the other hand, by the computational Diffie-Hellman (CDH) problem it is hard to compute σ from g, h, m. Such "gap Diffie-Hellman" groups can be instantiated from bilinear maps.

The only thing that remains is to guarantee that public keys are random bit strings. Since known constructions of bilinear maps are just special elliptic curve groups, we can apply the same modifications as we did for key agreement above to get random public keys.

7 Proof of Theorem 5

In this section, we give the full description of our simulator and the rest proof of Theorem 5.

Simulator In Ideal Game. Let $(\mathsf{KEYGEN}, \mathsf{SHAREDKEY})$ be the function pair that samples from $\mathcal{T}_{NIKE}$, the simulator works as follows.

H_0-Query. Suppose (x, H_0) is the k-th query($k \in [1, q]$), the system responds as follows:

- Case 1. If $Q_k \in H_0$, then it responds with the corresponding output string;
- Case 2. If there is a tuple $T = (T_1, T_2, T_3, T_4)$ in P-table, such that $T_4 = \mathsf{KEYGEN}(x)$ and $T_2 \neq *$, then the system responds with T_2 and inserts (x, T_2, T_3, T_4) into H_0 table.
- Case 3. Otherwise, the system samples $r \leftarrow \mathcal{X}$, responds to (x, H_0) by r and inserts the tuple $(x, r, \mathsf{keygen}(r), \mathsf{KEYGEN}(x))$ and $(*, r, \mathsf{keygen}(r), \mathsf{KEYGEN}(x))$ into the H_0 table and P-table, respectively.

P-Query. Suppose (y, P) be the k-th query, the system responds as follows:

- Case 1. If $Q_k \in H_0 \cup P \cup P^{-1}$, then responds with the third element of the 4-tuple corresponding to Q_k;
- Case 2. Otherwise, it samples $r \leftarrow \mathcal{X}$, computes $\mathsf{keygen}(r)$. The simulator responds to the query by $\mathsf{keygen}(r)$ and inserts $(*, r, \mathsf{keygen}(r), y)$ into table for P.

P^{-1}-QUERY. Suppose (y, P) be the k-th query, the system responds as follows:

- Case 1. If $Q_k \in H_0 \cup P \cup P^{-1}$, then responds with the fourth element of the 4-tuple corresponding to Q_k;
- Case 2. Otherwise, it responds with a random string $y' \leftarrow \mathcal{Y}$ and inserts $(*, *, y, y')$ into P^{-1}-table.

H_1-QUERY. Suppose (y_1, y_2, z, H_1) be the k-th query, the system responds as follows:

- Case 1: If $Q_k \in H_1$, then responds the corresponding string;
- Case 2: If $Q_k \notin H_1$ and $\mathsf{WCC}_k = 1$, then the system would test the validity of z only using P and P^{-1} table. Formally,
  1. If $y_1 > y_2$, then responds with $\underline{\text{a random string}}$;
  2. If $y_1 \leq y_2$ and there is a tuple $T = (T_1, T_2, T_3, T_4)$ in P^{-1}-table such that $T_4 = y_1$ and $T_1 \neq *$, then searches every tuple $T' = (T_1', T_2', T_3', T_4')$ in P-table which satisfies $T_4' = y_2$ and $T_2' \neq *$. If the tuple T' is found, then tests if $z \overset{?}{=} \mathsf{sharedkey}(T_3, T_2')$(only using the tuples in $P \cup P^{-1}$). If so, responds with $\mathsf{NSK}(T_4', T_1)$, otherwise responds with $\underline{\text{a random string}}$;
  3. If $y_1 \leq y_2$ and there is a tuple $T = (T_1, T_2, T_3, T_4)$ in P^{-1}-table such that $T_4 = y_2$ and $T_1 \neq *$, then searches every tuple $T' = (T_1', T_2', T_3', T_4')$ in P-table which satisfies $T_4' = y_1$ and $T_2' \neq *$. If the tuple T' is found, then test if $z \overset{?}{=} \mathsf{sharedkey}(T_3, T_2')$(only using the tuples in $P \cup P^{-1}$). If so, responds with $\mathsf{NSK}(T_4', T_1)$, otherwise responds with $\underline{\text{a random string}}$;
  4. Otherwise, responds with $\underline{\text{a random string}}$.
- Case 3: If $Q_k \notin H_1$ and $\mathsf{WCC}_k = 0$,
  1. Suppose that there is some query $Q_i, i < k$ to H_0 which contains an x such that $\mathsf{KEYGEN}(x) = y_1$.
     Suppose there is no such query for y_2. Then searches the P and P^{-1} tables for the value y_2 in the 4th position, corresponding to an input to P. If no such tuple is found, respond to the query with $\underline{\text{a random string}}$.
     If a tuple is found, let T be the 3rd element in the tuple, corresponding to the output of P on y_2. Then test if $z \overset{?}{=} \mathsf{sharedkey}(T, \tilde{H}_0(x))$. If so, respond with $\mathsf{SHAREDKEY}(y_2, x)$, otherwise respond with $\underline{\text{a random string}}$;
  2. The case where there is a query Q_i for y_2 but y_1 is handled analogously.
  3. If both have preimages in the H_0 table, say $\mathsf{KEYGEN}(x_1) = y_1$, $\mathsf{KEYGEN}(x_2) = y_2$, where x_1, x_2 are included in previous queries, then the system would test $z \overset{?}{=} \mathsf{sharedkey}(\mathsf{keygen}(H_0(x_i)), H_0(x_j))$ (using the tuples in H_0-table). If valid, then the system responds with $\mathsf{SHAREDKEY}(y_2, x_1)$, else $\underline{\text{a random string}}$.

Note that our simulator works the same as the system in Game 10. Moreover, the distribution of the honest interfaces in Game 10 is also identical to the ones in the ideal game, which means, for any adversary, the advantage in both Game 10 and ideal game is identical. Therefore, it suffices to prove that any adjacent games are indistinguishable.

Before that, we give two useful lemmas based on security.

Lemma 12. *Let* $X := \mathsf{keygen}(r)$ *be a variable, where* $r \leftarrow \mathcal{X}$*, then* $\forall y \in \mathcal{Y}$*,* $\Pr[X = y] \leq \sqrt{2\epsilon_3 + \frac{1}{|\mathcal{X}|}}$.

Proof. Note that the advantage of pseudorandom public key is ϵ_3, by hybrid argument, it's trivial that

$$\mathrm{Adv}_{\mathcal{A}} := |\Pr[\mathcal{A}(\mathsf{keygen}(r_1), \mathsf{keygen}(r_2))] - \Pr[\mathcal{A}(R_1, R_2)]| \leq 2\epsilon_3$$

where $r_1, r_2 \leftarrow \mathcal{X}, R_1, R_2 \leftarrow \mathcal{Y}$.

Assuming there exists y^* such that $\Pr[X = y] > \sqrt{2\epsilon_3 + \frac{1}{|\mathcal{X}|}}$, then $\Pr[\mathsf{keygen}(r_1) = \mathsf{keygen}(r_2)] > 2\epsilon_3 + \frac{1}{|\mathcal{X}|}$. On the other hand we have $\Pr[R_1 = R_2] \leq \frac{1}{|\mathcal{X}|}$, which means we can differ the tuple with a better advantage than $2\epsilon_3$.

Similar to Lemma 12 , we have the following lemma.

Lemma 13. *Let* $\mathsf{sk} \in \mathcal{X}$*, and* Z *be a variable that* $Z := \mathsf{sharedkey}(\mathsf{sk}, \mathsf{keygen}(r))$ *where* $r \leftarrow \mathcal{X}$*, then* $\forall z \in \mathcal{Z}$*,* $\Pr[Z = z] \leq \sqrt{2\epsilon_2 + \frac{1}{|\mathcal{Z}|}}$.

Now, we are ready to prove the indistinguishability between the games.

Claim. Game 0 $\approx$ Game 1.

Proof. It's trivial to note that Pr[Game 0] = Pr[Game 1], as in Game 1, the system only changes the way of maintaining the tables and the terms stored in each tuple is identical to the responses of real oracles.

Claim. Game 1 $\approx$ Game 2.

Proof. Firstly, we note that, in either Game 1 or Game 2, the adversary's view on the honest interfaces is consistent; hence it suffices to prove that, with high probability, the view on the adversarial interfaces also does not change. According to the definition, the only difference between Game 1 and Game 2 is when Case 3 occurs. Suppose (y, P) is the k-th query, the system sets $P(\mathsf{y}) = \mathsf{r}$ where $\mathsf{r} \leftarrow \mathcal{Y}$. Then the adversary's views on the adversarial interfaces are as follows:

View on H_0. By definition, $\tilde{H_0}$ is a random oracle such that 1) the response is consistent; 2) the response to a fresh input is uniformly random.

Thus, the adversary's view on H_0 is consistent in both games, if the responses of H_0 queries satisfy the two properties above. When case 3 occurs, the responses of H_0 queries do not change as it always responds with $\tilde{H_0}$.

View on P. By definition, $\tilde{P}$ is a random permutation oracle such that 1) the response is consistent; 2) the response to a fresh input is uniformly random; 3) the response is injective.

Thus, the adversary's view on P is consistent in both games, if the responses of P queries satisfy the three properties above. When case 3 occurs, after Q_k, the system sets $P(\mathsf{y}) = \mathsf{r}$. As r is randomly chosen, property 1, 2 hold trivially, and property 3 holds if there is no collision of P queries on r.

View on P^{-1}. By definition $\tilde{P^{-1}}$ is the inverse of a random permuatation oracle such that 1) the response is consistent; 2) the response is the inverse of P; 3) the response is injective; 4) $P^{-1}(\mathsf{keygen}(H_0(\mathsf{x}))) = \mathsf{NKG}(\mathsf{x})$.

Thus, the adversary's view on P^{-1} is consistent in both games, if the responses of P queries satisfy the four properties above. When case 3 occurs, after Q_k, the system sets

$P^{-1}(r) = y$, which implies property 2. Moreover, if r never appears in the previous queries, and $\tilde{P}(\mathsf{y}), \widetilde{P^{-1}}(\mathsf{r})$ never appear during the queries, then property 1 and 3 also hold. For property 4, And easy to note that property 4 holds if the adversary cannot output x^* such that $\mathsf{keygen}(\tilde{H}_0(x^*)) = \tilde{P}(y)$ or $\mathsf{keygen}(\tilde{H}_0(x^*)) = \mathsf{r}$. Applying Lemma 12, this bad event is bounded by $2q\sqrt{2\epsilon_3 + \frac{1}{|\mathcal{X}|}}$.

View on H_1. By definition, $\tilde{H}_1$ is a random oracle such that 1) the response is consistent 2) the response to a fresh input is uniformly random; 3) the response is injective; and

4) $H_1(P^{-1}(\mathsf{keygen}(\mathsf{r})), \mathsf{NKG}(\mathsf{x}), \mathsf{sharedkey}(P(\mathsf{NKG}(\mathsf{x})), r)) = \mathsf{NSK}(P^{-1}(\mathsf{keygen}(\mathsf{r})), x)$

5) $H_1(\mathsf{NKG}(\mathsf{x}), \mathsf{y}, \mathsf{sharedkey}(P(\mathsf{y}), H_0(\mathsf{x}))) = \mathsf{NSK}(\mathsf{y}, \mathsf{x})$

6) $H_1(\mathsf{NKG}(\mathsf{x}_1), \mathsf{NKG}(\mathsf{x}_2), \mathsf{sharedkey}(\mathsf{keygen}(H_0(\mathsf{x}_1)), H_0(\mathsf{x}_2))) = \mathsf{NSK}(\mathsf{NKG}(\mathsf{x}_1), \mathsf{x}_2) = \mathsf{NSK}(\mathsf{NKG}(\mathsf{x}_2), \mathsf{x}_1)$

(here we always assume $\mathsf{PK}_1 \leq \mathsf{PK}_2$).

Thus, the adversary's view on H_1 is consistent in both games, if the responses of P queries satisfy the six properties above. As in the view on P^{-1}, we've already assumed that x^* such that $\mathsf{NKG}(\mathsf{x}) = \mathsf{y}$ and $\tilde{P}(y)$ that never appear by giving a bound, property 4, 5 and 6 hold automatically. And if no collision, property holds. Moreover, when case 3 occurs, the system sets

$$H_1(\mathsf{NKG}(\mathsf{x}), \mathsf{y}, \mathsf{sharedkey}(r, H_0(\mathsf{x}))) = \mathsf{NSK}(\mathsf{y}, \mathsf{x}).$$

If $\mathsf{sharedkey}(r, *))$, $\mathsf{sharedkey}(\tilde{P}(\mathsf{y}), *))$ and $\mathsf{sharedkey}(*, \tilde{H}_0(\mathsf{x}))$ such that $\mathsf{NKG}(\mathsf{x}) = y$ or $\mathsf{keygen}(\tilde{H}_0(\mathsf{x})) = \mathsf{r}$ never appear, then the response is consistent and $\mathsf{NSK}(y, x)$ is a random value, as

$$\mathsf{NSK}(\mathsf{y}, \mathsf{x}) = \tilde{H}_1(\mathsf{NKG}(\mathsf{x}), \mathsf{y}, \mathsf{sharedkey}(P(\mathsf{y}), H_0(\mathsf{x}))).$$

Due to Lemma 13, this event can be bounded by $4q\sqrt{2\epsilon_2 + \frac{1}{|\mathcal{Z}|}}$.

Now, we can bound the union of these bad events as:

$$\Pr[\mathsf{Bad}] \leq \frac{q}{|\mathcal{Y}|} + \frac{k-1}{|\mathcal{Y}|} + \frac{q-k}{|\mathcal{Y}|} + 2q\sqrt{2\epsilon_3 + \frac{1}{|\mathcal{X}|}} + \frac{q}{|\mathcal{W}|} + 4q\sqrt{2\epsilon_2 + \frac{1}{|\mathcal{Z}|}} \leq \mathsf{negl}(\lambda),$$

which referring to

$$|\Pr[\text{Game 1}] - \Pr[\text{Game 2}]| \leq q \cdot \Pr[\mathsf{Bad}] \leq \mathsf{negl}(\lambda).$$

Claim. Game 2 $\approx$ Game 3.

Proof. This proof is trivial as we just change the distribution of outputs from random strings to random public key. Hence, due to the pseudorandomness of keygen, we have

$$|\Pr[\text{Game 2}] - \Pr[\text{Game 3}]| \leq q\epsilon_3 \leq \mathsf{negl}(\lambda).$$

Claim. Game 3 $\approx$ Game 4.

Proof. In Game 4, the system only changes the way of maintaining the tables, and if r is not used, then the view in both games is consistent. Thus,

$$|\Pr[\text{Game 3}] = \Pr[\text{Game 4}]| \leq |\Pr[\text{Game 1}] = \Pr[\text{Game 2}]|.$$

Claim. Game 4 $\approx$ Game 5.

Proof. Firstly, we note that, in either Game 4 or Game 5, the adversary's view on the honest interfaces is consistent, hence it suffices to prove that, with high probability, the view on the adversarial interfaces also does not change. According to the definition, the only difference between Game 4 and Game 5 is when Case 3 occurs. Suppose (y, P) is the k-th query, where $\mathsf{y} = \mathsf{NKG}(\mathsf{x})$ and x is known by the adversary, then its views on the adversarial interfaces are as follows:

View on H_0. By definition, $\tilde{H_0}$ is a random oracle such that 1) the response is consistent; 2) the response to a fresh input is uniformly random.

Thus, the adversary's view on H_0 is consistent in both games, if the responses of H_0 queries satisfy the two properties above. When case 3 occurs, the system implicitly sets $H_0(\mathsf{x}) = \mathsf{r}$, as r is randomly chosen, and (x, H_0) never appears in the previous queries, property 1 and 2 hold trivially.

View on P. By definition, $\tilde{P}$ is a random permutation oracle such that 1) the response is consistent; 2) the response to a fresh input is uniformly random; 3) the response is injective.

Thus, the adversary's view on P is consistent in both games, if the responses of P queries satisfy the three properties above. When case 3 occurs, after Q_k, the system sets $P(\mathsf{y}) = \mathsf{keygen}(r), \mathsf{r} \leftarrow \mathcal{X}$. Due to the pseudorandomness of the public key, property 1, 2 and 3 preserve.

View on P^{-1}. By definition $\tilde{P^{-1}}$ is the inverse of a random permuatation oracle such that 1) the response is consistent; 2) the response is the inverse of P; 3) the response is injective; 4) $P^{-1}(\mathsf{keygen}(H_0(\mathsf{x}))) = \mathsf{NKG}(\mathsf{x})$.

Thus, the adversary's view on P^{-1} is consistent in both games, if the responses of P queries satisfy the four properties above. When case 3 occurs, after Q_k, the system sets $P^{-1}(\mathsf{keygen}(\mathsf{r})) = \mathsf{y}$, which implies property 2 and 4. If $\mathsf{keygen}(\mathsf{r})$ never appears in the previous queries, and $\tilde{P}(\mathsf{y})$, $\tilde{H_0}(\mathsf{x})$ and $\tilde{H_0}(\mathsf{x}^*)$ such that $\mathsf{keygen}(\mathsf{x}^*) = \mathsf{r}$ never appear, then property 1 and 3 also hold. By lemma 12, the probability of this event is bounded by $3q\sqrt{2\epsilon_3 + \frac{1}{|\mathcal{X}|}} + \frac{q}{|\mathcal{X}|}$.

View on H_1. By definition, $\tilde{H_1}$ is a random oracle such that 1) the response is consistent 2) the response to a fresh input is uniformly random; 3) the response is injective,

4) $H_1(P^{-1}(\mathsf{keygen}(\mathsf{r})), \mathsf{NKG}(\mathsf{x}), \mathsf{sharedkey}(P(\mathsf{NKG}(\mathsf{x})), r)) = \mathsf{NSK}(P^{-1}(\mathsf{keygen}(\mathsf{r})), x)$

5) $H_1(\mathsf{NKG}(\mathsf{x}), \mathsf{y}, \mathsf{sharedkey}(P(\mathsf{y}), H_0(\mathsf{x}))) = \mathsf{NSK}(\mathsf{y}, \mathsf{x})$

6) $H_1(\mathsf{NKG}(\mathsf{x}_1), \mathsf{NKG}(\mathsf{x}_2), \mathsf{sharedkey}(\mathsf{keygen}(H_0(\mathsf{x}_1)), H_0(\mathsf{x}_2))) = \mathsf{NSK}(\mathsf{NKG}(\mathsf{x}_1), \mathsf{x}_2) = \mathsf{NSK}(\mathsf{NKG}(\mathsf{x}_2), \mathsf{x}_1)$

Thus, the adversary's view on H_1 is consistent in both games, if the responses of P queries satisfy the five properties above. It's trivial that property 4, 5, 6 hold, as after Q_k the system implicitly sets those equations. Property 3 holds if no collision. Besides, if $\mathsf{keygen}(\mathsf{r})$ and $\mathsf{sharedkey}(\mathsf{keygen}(\mathsf{r}), *)$ never appear in the previous queries, then the response is consistent. However in Game 5, we note that, for any $\mathsf{x}', \mathsf{T} \in \mathcal{X}$,

$$H_1(P^{-1}(\mathsf{keygen}(\mathsf{T})), y, \mathsf{sharedkey}(\mathsf{keygen}(\mathsf{r}), \mathsf{T})) = H_1(P^{-1}(\mathsf{keygen}(\mathsf{T})), y, \mathsf{sharedkey}(\tilde{P}(y), \mathsf{T}))$$

$$H_1(\mathsf{NKG}(\mathsf{x}'), \mathsf{y}, \mathsf{sharedkey}(P(\mathsf{y}), H_0(\mathsf{x}'))) = H_1(\mathsf{NKG}(\mathsf{x}'), \mathsf{y}, \mathsf{sharedkey}(\tilde{P}(\mathsf{y}), H_0(\mathsf{x}')))$$

$$H_1(\mathsf{NKG}(\mathsf{x}'), y, \mathsf{sharedkey}(\mathsf{keygen}(\mathsf{r}), H_0(\mathsf{x}'))) = H_1(\mathsf{NKG}(\mathsf{x}'), y, \mathsf{sharedkey}(\mathsf{keygen}(\tilde{H_0}(\mathsf{x})), H_0(\mathsf{x}')))$$

which might break property 2. In fact, if $\mathsf{sharedkey}(\mathsf{r}, *)$, $\mathsf{sharedkey}(\tilde{P}(\mathsf{y}), *)$ and $\mathsf{sharedkey}(\mathsf{keygen}(\tilde{H}_0(*, \mathsf{x}^*)))$ ($\mathsf{keygen}(\mathsf{x}^*) = \mathsf{r}$ or $\mathsf{NKG}(\mathsf{x}^*) = \mathsf{y}$) never appear during the queries, then the responses are fresh and uniform. As in P^{-1} view, we've already assumed that $\tilde{P}(\mathsf{y})$ and $\tilde{H}_0(\mathsf{x})$ never appear, applying Lemma 13, we can bound the probability as $4q\sqrt{2\epsilon_2 + \frac{1}{|\mathcal{Z}|}}$.

Now, we can bound the union of these bad events as:

$$\Pr[\mathsf{Bad}] \leq q\epsilon_3 + 3q\sqrt{2\epsilon_3 + \frac{1}{|\mathcal{X}|}} + \frac{q}{|\mathcal{X}|} + \frac{q}{|\mathcal{W}|} + 4q\sqrt{2\epsilon_2 + \frac{1}{|\mathcal{Z}|}} \leq \mathsf{negl}(\lambda)$$

which referring to

$$|\Pr[\text{Game 4}] - \Pr[\text{Game 5}]| \leq q \cdot \Pr[\mathsf{Bad}] \leq \mathsf{negl}(\lambda).$$

Claim. Game 5 $\approx$ Game 6.

Proof. Firstly, we note that, in either Game 5 or Game 6, the adversary's view on the honest interfaces is consistent, hence it suffices to prove that, with high probability, the view on the adversarial interfaces also does not change. According to the definition, the only difference between Game 4 and Game 5 is when Case 2 occurs. Suppose (y, P^{-1}) is the k-th query and $\mathsf{SCC}_k = 1$, in Game 5 the system sets $P^{-1}(\mathsf{y}) = \mathsf{NKG}(\mathsf{x})$ where $\mathsf{x} \xleftarrow{\$} \mathcal{X}$, we note that adversary might know T such that $\mathsf{keygen}(\mathsf{T}) = \mathsf{y}$. Then the adversary's views on the adversarial interfaces are as follows:

View on H_0. In either Game 5 or Game 6, the responses of H_0 queries do not change.

View on P. By definition, $\tilde{P}$ is a random permutation oracle such that 1) the response is consistent; 2) the response to a fresh input is uniformly random; 3) the response is injective.

Thus, the adversary's view on P is consistent in both games, if the responses of P queries satisfy the three properties above. We note that, in both games $\tilde{P}$ is hidden and P-queries are answered by other tables and the honest interfaces, hence except negligible collision, property 3 holds.

View on P^{-1}. By definition $\tilde{P^{-1}}$ is the inverse of a random permuatation oracle such that 1) the response is consistent; 2) the response is the inverse of P; 3) the response is injective; 4) $P^{-1}(\mathsf{keygen}(H_0(\mathsf{x}))) = \mathsf{NKG}(\mathsf{x})$.

Thus, the adversary's view on P^{-1} is consistent in both games, if the responses of P queries satisfy the four properties above. When case 2 occurs, after Q_k, the system sets

$$P^{-1}(\mathsf{y}) = \mathsf{NKG}(x) = \tilde{P^{-1}}(\mathsf{keygen}(\tilde{H}_0(\mathsf{x})))$$

If $\mathsf{keygen}(\tilde{H}_0(\mathsf{x}))$ never appears in the previous queries, then $\mathsf{NKG}(x)$ is uniformly fresh, which means property 1, 2, 4 hold. For property 3, we note that if no x^*, such that $\mathsf{NKG}(\mathsf{x}^*) = \mathsf{NKG}(\mathsf{x})$ or $\mathsf{keygen}(\tilde{H}_0(\mathsf{x}^*)) = \mathsf{T}$ appears, then the responses are injective. Applying Lemma 12, we can bound it by $3q\sqrt{2\epsilon_3 + \frac{1}{|\mathcal{X}|}}$.

View on H_1. By definition, $\tilde{H}_1$ is a random oracle such that 1) the response is consistent 2) the response to a fresh input is uniformly random; 3) the response is injective,

4) $H_1(P^{-1}(\mathsf{keygen}(\mathsf{r})), \mathsf{NKG}(\mathsf{x}), \mathsf{sharedkey}(P(\mathsf{NKG}(\mathsf{x})), r)) = \mathsf{NSK}(P^{-1}(\mathsf{keygen}(\mathsf{r})), x)$

5) $H_1(\mathsf{NKG}(\mathsf{x}), \mathsf{y}, \mathsf{sharedkey}(P(\mathsf{y}), H_0(\mathsf{x}))) = \mathsf{NSK}(\mathsf{y}, \mathsf{x})$

6) $H_1(\mathsf{NKG}(\mathsf{x}_1), \mathsf{NKG}(\mathsf{x}_2), \mathsf{sharedkey}(\mathsf{keygen}(H_0(\mathsf{x}_1)), H_0(\mathsf{x}_2))) = \mathsf{NSK}(\mathsf{NKG}(\mathsf{x}_1), \mathsf{x}_2) = \mathsf{NSK}(\mathsf{NKG}(\mathsf{x}_2), \mathsf{x}_1)$

Thus, the adversary's view on H_1 is consistent in both games, if the responses of H_1 queries satisfy the five properties above. It's trivial that property 5, 6 hold, as after Q_k the system implicitly sets those equations. For property 4, for any $\mathsf{x}' \in \mathcal{X}$, the system sets as follows:

$$H_1(P^{-1}(\mathsf{keygen}(\mathsf{T})), \mathsf{NKG}(\mathsf{x}'), \mathsf{sharedkey}(P(\mathsf{NKG}(\mathsf{x}')), T)) = \mathsf{NSK}(\mathsf{NKG}(\mathsf{x}'), x) = \mathsf{NSK}(P^{-1}(\mathsf{keygen}(\mathsf{T})), x')$$

which means property 4 also holds. And if $\mathsf{NKG}(\mathsf{x})$ never appears in the previous query, then the response is consistent. However in Game 6, we note that, for any $\mathsf{x}' \in \mathcal{X}$,

$$\begin{aligned}&H_1(P^{-1}(\mathsf{keygen}(\mathsf{T})), \mathsf{NKG}(\mathsf{x}'), \mathsf{sharedkey}(P(\mathsf{NKG}(\mathsf{x}')), T))\\ =&H_1(P^{-1}(\mathsf{keygen}(\mathsf{T})), \mathsf{NKG}(\mathsf{x}'), \mathsf{sharedkey}(P(\mathsf{NKG}(\mathsf{x}')), \tilde{H_0}(\mathsf{x})))\end{aligned}$$

which might break property 2. In fact, if $\mathsf{sharedkey}(\tilde{P^{-1}}(\mathsf{y}), *)$, $\mathsf{sharedkey}(*, \tilde{H_0}(x^*))$ such that $\mathsf{DKG}(\mathsf{x}^*) = \mathsf{y}$ or $\mathsf{sharedkey}(\tilde{H_0}(\mathsf{x}^*)) = P(\mathsf{y})$ never appears during the queries, then the responses are fresh and uniform. Applying Lemma 13, we can bound the probability as $4q\sqrt{2\epsilon_2 + \frac{1}{|\mathcal{Z}|}}$.

Now, we can bound the union of these bad events as:

$$\Pr[\mathsf{Bad}] \leq \frac{q}{|\mathcal{Y}|} + 3q\sqrt{2\epsilon_3 + \frac{1}{|\mathcal{X}|}} + 4q\sqrt{2\epsilon_2 + \frac{1}{|\mathcal{Z}|}} + \frac{q}{|\mathcal{W}|} \leq \mathsf{negl}(\lambda),$$

which referring to

$$|\Pr[\text{Game 5}] - \Pr[\text{Game 6}]| \leq q \cdot \Pr[\mathsf{Bad}] \leq \mathsf{negl}(\lambda).$$

Claim. Game 6 $\approx$ Game 7.

Proof. Suppose $Q_k = (\mathsf{y}, P^{-1})$ is the k-th query and we denote event Bad as $\mathsf{SCC_k} = 0 \cap \mathsf{WCC_k} = 1$. Next, we bound the bad event.

Assuming Bad occurs, then there is a x known by the adversary such that $y = \mathsf{keygen}(\tilde{H_0}(\mathsf{x}))$ and (x, H_0) never appears in previous $k-1$ queries. As $\tilde{H_0}(\mathsf{x})$ is unifromly distributed in $\mathcal{X}$, the probability that it can output a proper y is bounded by the pseudorandomness of the public key. By definition, it's trivial to note that

$$|\Pr[\text{Game 6}] - \Pr[\text{Game 7}]| \leq q \Pr[\mathsf{Bad}] \leq q\epsilon_3 \leq \mathsf{negl}(\lambda).$$

Claim. Game 7 $\approx$ Game 8.

Proof. Firstly, we note that, in either Game 7 or Game 8, the adversary's view on the honest interfaces is consistent, hence it suffices to prove that, with high probability, the view on the adversarial interfaces also does not change. According to the definition, the only difference between Game 7 and Game 8 is when Case 3 occurs. Suppose (x, H_0) is the k-th query, in Game 8 the system sets $H_0(\mathsf{x}) = \mathsf{r}$ where $\mathsf{r} \xleftarrow{\$} \mathcal{X}$. While we note that adversary might know $\mathsf{NKG}(\mathsf{x})$ before Q_k, with the restriction that $\mathsf{NKG}(\mathsf{x})$ never appears in $P \cup P^{-1}$ in the previous queries. Then the adversary's views on the adversarial interfaces are as follows:

View on H_0. By definition, $\tilde{H_0}$ is a random oracle such that 1) the response is consistent; 2) the response to a fresh input is uniformly random.

Thus, the adversary's view on H_0 is consistent in both games, if the responses of H_0 queries satisfy the two properties above. When case 3 occurs, the system implicitly sets $H_0(\mathsf{x}) = \mathsf{r}$, as r is randomly chosen, and (x, H_0) never appears in the previous queries, property 1 and 2 hold trivially.

View on P and P^{-1}. In both games, P and P^{-1} are answered by H_0 table and the honest interfaces. As the view on H_0 does not change, except with negligible collision, the view on P and P^{-1} preserves.

View on H_1. In both games, it is trivial that those equations hold trivially, and same as above, if $\mathsf{sharedkey}(\mathsf{r}, *)$, $\mathsf{sharedkey}(, \tilde{H}_0(\mathsf{x}))$ and $\mathsf{sharedkey}(, \tilde{H}_0(\mathsf{x}^*))$ such that $\tilde{H}_0(x^*) = \mathsf{r}$ never appear during the queries, then property 1 and 2 hold. And applying Lemma 13, we can we can bound it by $3q\sqrt{2\epsilon_2 + \frac{1}{|\mathcal{Z}|}}$.

Now, we can bound the union of these bad events as:

$$\Pr[\mathsf{Bad}] \leq \frac{q}{|\mathcal{Y}|} + \frac{q}{|\mathcal{W}|} + 3q\sqrt{2\epsilon_2 + \frac{1}{|\mathcal{Z}|}} \leq \mathsf{negl}(\lambda)$$

which referring to

$$|\Pr[\text{Game } 7] - \Pr[\text{Game } 8]| \leq q \cdot \Pr[\mathsf{Bad}] \leq \mathsf{negl}(\lambda).$$

Claim. Game 8 $\approx$ Game 9.

Proof. Firstly, we note that, in either Game 8 or Game 9, the adversary's view on the honest interfaces is consistent, hence it suffices to prove that, with high probability, the view on the adversarial interfaces also does not change. According to the definition, we note that in Game 9, the system replaces the responses, which fails in the check, with a random string.

View on H_0, P, P^{-1}. It's trivial that, in either Game 8 or Game 9, the H_0, P, P^{-1} queries are responded in the same way, hence the view does not change.

View on H_1. We note that in both games, $\tilde{H}_0, \tilde{P}, \tilde{P^{-1}}$ has been completely hidden, hence the probability that $\mathcal{A}$ outputs $(\mathsf{y}_1, \mathsf{y}_2, \mathsf{z})$ such that z is a valid sharedkey under $\tilde{H}_0, \tilde{P}, \tilde{P^{-1}}$ is bounded by pseudorandomness of the sharedkey. Now, it's rest to show that, with high probability, for the query (y_1, y_2, z) which fails the consistency check, z is also not a valid shared key for y_1, y_2 under H_0, P, P^{-1}. It's trivial to rule out the cases where the shared-key test fails, and for the rest cases, we observe that, the adversary is obligated to output a valid shared key either without knowing the two secret keys or knowing one secret key but nothing of the other public key. This bad event is bounded by the security properties: *unpredictable shared key* and *pseudorandom shared key.* Thus, we have

$$|\Pr[\text{Game } 8] - \Pr[\text{Game } 9]| \leq 2q\sqrt{2\epsilon_2 + \frac{1}{|\mathcal{Z}|}} + q\epsilon_1 \leq \mathsf{negl}(\lambda).$$

Claim. Game 9 $\approx$ Game 10.

Proof. Due to definition, we note that in Game 9, the system responds to all of the adversarial interfaces just using tables and honest interfaces, it never directly calls the real oracles. Besides, $\tilde{H}_0, \tilde{H}_1$ are random oracles and $\tilde{P}, \tilde{P^{-1}}$ are random permutations, hence the only chance the adversary can differ Game 9 and Game 10 is leading to a collision, on either KEYGEN or $\mathsf{SHAREDKEY}$. Hence

$$|\Pr[\text{Game } 9] - \Pr[\text{Game } 10]| \leq q^2\sqrt{2\epsilon_3 + \frac{1}{|\mathcal{X}|}} + \frac{q^2}{|\mathcal{W}|} \leq \mathsf{negl}(\lambda).$$

Combining together, we complete the whole proof. □

8 Proof of Theorem 7

In this section, we give the full proof of Theorem 7, that we show Π_{DPKE} is indifferentiable from an ideal DPKE.

Proof. According to the definition of indifferentiability, we immediately observe that, the adversary has three honest interfaces ($\mathsf{DKG}, \mathsf{DE}, \mathsf{DD}$) and six adversarial interfaces (H_0, H_1, P,P^{-1}, $\mathsf{NKG}, \mathsf{NSK}$). Therefore, we need to build an efficient simulator $\mathcal{S}$ that can simulate the six adversarial interfaces $H_0, H_1, P, \mathsf{NKG}$ and NSK properly, which means, for any PPT differentiator $\mathcal{D}$, the view of $\mathcal{D}$ in the real game is computationally close to the view in the ideal game. To do so, we will go through with a sequence of hybrid games. Before the description of the games, we first specify some parameters:

- there are nine types of query such as: $(x, H_0), (y, m, H_1), (y, m, P), (z, P^{-1})$, $(r, \mathsf{NKG}), (y, r, \mathsf{NSK}), (x, \mathsf{DKG}), (y, m, \mathsf{DE}), (y, z, x, \mathsf{DD})$ where $x, r \leftarrow \mathcal{X}, y \leftarrow \mathcal{Y}$, $z \leftarrow \mathcal{Z}$;
- adversary makes at most q queries to the system, where $q = \mathsf{poly}(\lambda)$;
- the real oracles used in the construction are $\tilde{H}_0, \tilde{H}_1, \tilde{P}, \tilde{P}^{-1}, \tilde{\mathsf{NKG}}, \tilde{\mathsf{NSK}}$.

Now we are ready the define the games.

Game 0. This game is identical to the real game, where every query is answered by applying real oracles. For instance, the response of (x, H_0) is $\tilde{H}_0(x)$, the response of (x, DKG) is $\tilde{\mathsf{NKG}}(\tilde{H}_0(x)))$ and so forth. Moreover, during the queries, it also maintains six tables, referring to H_0-table, H_1-table, P-table, P^{-1}-table, NKG-table and NSK-table. Concretely,

- H_0-table: Initially empty, consists of tuples with form of (x, r, y). Once the adversary makes a (x, H_0) query which does not exist in H_0-table (no tuple that the first element of it is x), it inserts $(x, \tilde{H}_0(x), \mathsf{DKG}(x))$ into the table.
- H_1-table: Initially empty, consists of tuples with form of (y, m, r, y'). Once the adversary makes a (y, m, H_1) query that does not exist in H_1-table, it inserts $(y, m, \tilde{H}_1(y, m), \mathsf{DE}_1(y, m))$ into the table(here by $\mathsf{DE}_1(y, m)$, we mean the first element of $\mathsf{DE}(y, m)$, and below $\mathsf{DE}_2(y, m)$ is used analogously).
- P-table: Initially empty, consist of tuples with form of (y, m, z). Once the adversary makes a (y, m, P) query which does not exist in P-table, it inserts $(y, m, \tilde{P}(y, m))$ into the table.
- P^{-1}-table: Initially empty, consist of form of (y, m, z). Once the adversary makes a (z, P^{-1}) query which does not exist in P-table, it inserts $(\tilde{P^{-1}}(z), z)$ into the table(here we abuse $\tilde{P^{-1}}(z)$ to be 2 elements, (y, m)).
- NKG-table. Initially empty, consist of tuples with form of $(*, r, y)$. Once the adversary makes a (r, NKG) query which does not exist in NKG-table, it inserts $(*, r, \tilde{\mathsf{NKG}}(r))$ into the table.

– NSK-table. Initially empty, consist of tuples with form of (y, x, z). Once the adversary makes a (y, x, NKG) query which does not exist in NKG-table, it inserts $(y, x, \tilde{\mathsf{NKG}}(y, x))$ into the table.

Note that at this point these tables are just keeping track of information relating to adversary's queries, and are completely hidden to the adversary. Next, we define a relation based on these tables. We will say a H_0 query $Q_k = (x, H_0)$ is in a table K, denoted $Q_k \in K$, if there is a 3-tuple in K such that the 1st element is equal to x. Analogously, for a P query we say $Q_k = (y, m, P) \in K$ if there is a 3-tuple $T = (T_1, T_2, T_3)$ in K such that $T_1 = y, T_2 = m$; for P^{-1} queries, we say $Q_k = (z, P^{-1}) \in K$ if there is a 3-tuple in K such that the 3rd element is equal to z. For a H_1-query, we say $Q_k = (y, m, H_1) \in K$ if there is a 3-tuple $T = (T_1, T_2, T_3)$in K such that $T_1 = y, T_2 = m$. For a NKG query, we say $Q_k = (r, \mathsf{NKG}) \in K$ if there is a 3-tuple $T = (T_1, T_2, T_3)$ such that $T_2 = r$ or there is a 4-tuple $T = (T_1, T_2, T_3, T_4)$ such that $T_3 = r$. For NSK query, we say $Q_k = (y, r, \mathsf{NSK}) \in \mathsf{K}$ if there is a 3-tuple $T = (T_1, T_2, T_3)$ in K such that $T_1 = y, T_2 = r$.

Then, we show a game that the system responds to the queries by using both tables and the real oracles without changing adversary's view.

Game 1. This game is identical to Game 0, except the way of maintaining the tables and responding to the queries. Specifically,

H_0-QUERY. Suppose (x, H_0) is the k-th query, the system responds as follows:

– Case 1. If $Q_k \in H_0$, then it responds with the corresponding output string;
– Case 2. If $Q_k \notin H_0$ but there is a tuple $T = (T_1, T_2, T_3)$ in NKG-table such that $T_3 = \mathsf{DKG}(x)$, then responds with T_2 and inserts (x, T_2, T_3) into H_0 table;
– Case 3. If $Q_k \notin H_0$ but there is a tuple $T = (T_1, T_2, T_3, T_4)$ in H_1-table such that $T_4 = \mathsf{DKG}(x)$, then responds with T_3 and inserts (x, T_3, T_4) into H_0 table;
– Case 4. Otherwise, responds with $\tilde{H}_0(x)$ and inserts $(x, \tilde{H}_0(x), \mathsf{DKG}(x))$ into H_0-table.

H_1-QUERY.Suppose (y, m, H_1) is the k-th query, the system responds as follows:

– Case 1. If $Q_k \in H_1$, then it responds with the corresponding output string;
– Case 2. If $Q_k \notin H_1$ but there is a tuple $T = (T_1, T_2, T_3)$ in NKG-table such that $T_3 = \mathsf{DE}_1(y, m)$, then responds with T_2 and inserts (y, m, T_2, T_3) into H_1 table;
– Case 3. If $Q_k \notin H_1$ but there is a tuple $T = (T_1, T_2, T_3)$ in H_0-table such that $T_3 = \mathsf{DE}_1(y, m)$, then responds with T_2 and inserts (y, m, T_2, T_3) into H_1 table;
– Case 4. Otherwise, responds with $\tilde{H}_1(y, m)$ and inserts $(y, m, \tilde{H}_1(y, m), \mathsf{DE}_1(y, m))$ into H_1-table.

NKG-QUERY. Suppose (r, NKG) is the k-th query, the system responds as follows:

– Case 1. If $Q_k \in \mathsf{NKG}$, then it responds with the corresponding output string;
– Case 2. If $Q_k \notin \mathsf{NKG}$ but there is a tuple $T = (T_1, T_2, T_3)$ in H_0-table such that $T_2 = r$, then responds with T_3 and inserts (T_1, T_2, T_3) into NKG table;
– Case 3. If $Q_k \notin \mathsf{NKG}$ but there is a tuple $T = (T_1, T_2, T_3, T_4)$ in H_1-table such that $T_3 = r$, then responds with T_4 and inserts $(*, T_3, T_4)$ into NKG table;
– Case 4. Otherwise, responds with $\tilde{\mathsf{NKG}}(r)$ and inserts $(*, r, \tilde{\mathsf{NKG}}(r))$ into NKG-table.

P-QUERY. Suppose (y, m, P) is the k-th query, the system responds as follows:

- Case 1. If $Q_k \in P \cup P^{-1}$, then it responds with the corresponding output string;
  Note that for the cases below where $Q_k \notin P \cup P^{-1}$, after we compute the response z, we will always add the tuple (y, m, z) to P-table and P^{-1}-table, so that on future identical queries, Q_k will be in $P \cup P^{-1}$.
- Case 2. If $Q_k \notin P \cup P^{-1}$ but there is a tuple $T = (T_1, T_2, T_3, T_4)$ in H_1-table such that $T_4 = \mathsf{DE}_1(y, m)$, then the system searches every tuple $T' = (T'_1, T'_2, T'_3)$ in NSK-tabe such that $T'_1 = y, T'_2 = T_3$. If such a tuple T' is found, then the system responds to the query with $\mathsf{DE}_2(y, m) \oplus T'_3$, else the system checks the following cases.
- Case 3. If $Q_k \notin P \cup P^{-1}$ but there is a tuple $T = (T_1, T_2, T_3)$ in $H_0 \cup \mathsf{NKG}$ such that $T_3 = \mathsf{DE}_1(y, m)$, then the system searches every tuple $T' = (T'_1, T'_2, T'_3)$ in NSK-tabe such that $T'_1 = y, T'_2 = T_2$. If such a tuple T' is found, then the system responds to the query with $\mathsf{DE}_2(y, m) \oplus T'_3$, else the sytem checks the following cases.
- Case 4. If $Q_k \notin P$ but there is tuple $T = (T_1, T_2, T_3)$ in $H_0 \cup \mathsf{NKG}$ such that $T_3 = y$, then the system searches every tuple $T' = (T'_1, T'_2, T'_3)$ in NSK-table such that $T'_1 = \mathsf{DE}_1(y, m), T'_2 = T_2$. If such a tuple is found, then the system responds with $\mathsf{DE}_2(y, m) \oplus T'_3$, else checks the following cases.
- Case 5. If $Q_k \notin P$ but there is tuple $T = (T_1, T_2, T_3, T_4)$ such that $T_4 = y$, then the system searches every tuple $T' = (T'_1, T'_2, T'_3)$ in NSK-table such that $T'_1 = \mathsf{DE}_1(y, m), T'_2 = T_3$. If such a tuple is found, then the system responds with $\mathsf{DE}_2(y, m) \oplus T'_3$.
- Case 6. Otherwise, the system responds with $\tilde{P}(y, m)$.

NSK-QUERY. Suppose (y, r, NSK) is the k-th query, the system responds as follows:

- Case 1. If $Q_k \in \mathsf{NSK}$, then it responds with the corresponding output string;
- Case 2. If $Q_k \notin \mathsf{NSK}$ but there is a tuple $(T_1, T_2, T_3) \in \mathsf{NSK}$ such that $T_1 = \mathsf{NKG}(r), y = \mathsf{NKG}(T_2)$, then the system responds with T_3;
- Case 3. If $Q_k \notin \mathsf{NSK}$ but there is tuple $T = (T_1, T_2, T_3, T_4)$ in H_1-table such that $T_1 = y, T_3 = r$, then searches every tuple $T' = (T'_1, T'_2, T'_3) \in P \cup P^{-1}$ such that $T'_1 = y, T'_2 = T_2$. If such a tuple is found, then the system responds with $\mathsf{DE}_2(y, T_2) \oplus T'_3$, else checks the following cases;
- Case 4. If $Q_k \notin \mathsf{NSK}$ but there is a tuple $T = (T_1, T_2, T_3) \in H_0 \cup \mathsf{NKG}$ such that $T_2 = r$, then searches every tuples $T' = (T'_1, T'_2, T'_3) \in P \cup P^{-1}$ such that $\mathsf{DE}_1(T'_1, T'_2) = y$ and $T'_1 = T_3$. If such a tuple is found, then the system repsonds with $\mathsf{DE}_2(T'_1, T'_2) \oplus T'_3$, else checks the following cases.
- Case 5. Otherwise, the system responds with $\tilde{\mathsf{NSK}}(y, r)$.

P^{-1} QUERY. Suppose (z, P^{-1}) is the k-th query, the system responds as follows:

- Case 1. If $Q_k \in P \cup P^{-1}$, then it responds with the corresponding output string;
- Case 2. If $Q_k \in P \cup P^{-1}$ but there are a tuple $T = (T_1, T_2, T_3, T_4) \in H_1$ and $T' = (T'_1, T'_2, T'_3) \in \mathsf{NSK}$ such that $T_1 = T'_1, T_3 = T_2$ and $\mathsf{DE}_2 = z \oplus T'_3$. If such tuples are found, then the system responds with (T_1, T_2), else checks the following cases;
- Case 3. If $Q_k \notin P \cup P^{-1}$ but there are a tuple $T = (T_1, T_2, T_3) \in H_0 \cup \mathsf{NKG}$ and $T' = (T'_1, T'_2, T'_3)$ such that $T_1 \neq *, T_2 = T'_2$ and $\mathsf{DD}(T'_1, z \oplus T'_3, T_1) \neq \perp$, then the system responds with $(T_3, \mathsf{DD}(T'_1, z \oplus T'_3, T_1))$, else checks the following cases;
- Case 4. Otherwise, the system responds with $\widetilde{P^{-1}}(z)$.

Note that, in Game 1 the system keeps a longer table, and for each query, it firstly checks whether it can be responded by the tables or honest interfaces. If can, then it responds with them, else calls the real oracles. In fact, the system in Game 1 is a prototype for our simulator

$\mathcal{S}$, and in the following games, we replace the responses that answered by the real oracles with random strings step by step. Moreover, the tuples stored in the table are consistent with the real oracles, hence each response in Game 1 is identical to the one in Game 0.

Game 2. This game is identical to Game 1, except that to answer H_0 queries, where the system only uses the tables and honest interfaces. Assuming (x, H_0) is the k-th query, then

- Case 1. If $Q_k \in H_0$, then same as above;
- Case 2. If $Q_k \notin H_0$ but there is a tuple $T = (T_1, T_2, T_3)$ in NKG-table such that $T_3 = \mathsf{DKG}(x)$, then same as above;
- Case 3. If $Q_k \notin H_0$ but there is a tuple $T = (T_1, T_2, T_3, T_4)$ in H_1-table such that $T_4 = \mathsf{DKG}(x)$, then same as above;
- Case 4. Otherwise, the systems randomly samples $r \leftarrow \mathcal{X}$, responds with r and inserts $(x, r, \mathsf{DKG}(x))$ into H_0-table.

Recalling the H_0-table in Game 1, we immediately observe that, the only case that the system calls $\tilde{H}_0(x)$ is when the adversary knows nothing about $\tilde{H}_0(x)$, although the adversary might know $\mathsf{NKG}(x)$. Therefore, from the adversary's view, $\tilde{H}_0(x)$ is uniformly distributed in $\mathcal{X}$, and it is equivalent to randomly pick $r \leftarrow \mathcal{X}$ and implicitly set $r = (x, H_0)$ and $\mathsf{DKG}(x) = (r, \mathsf{NKG})$.

Game 3. This game is identical to Game 2, except that to answer H_1 queries, where the system only uses the tables and honest interfaces. Assuming (y, m, H_1) is the k-th query, then

- Case 1. If $Q_k \in H_1$, then same as above;
- Case 3. If $Q_k \notin H_1$ but there is a tuple $T = (T_1, T_2, T_3)$ in NKG-table such that $T_3 = \mathsf{DE}_1(y, m)$, then same as above;
- Case 2. If $Q_k \notin H_1$ but there is a tuple $T = (T_1, T_2, T_3)$ in H_0-table such that $T_3 = \mathsf{DE}_1(y, m)$, then same as above;
- Case 4. Otherwise, the systems randomly samples $r \leftarrow \mathcal{X}$, responds with r and inserts $(y, m, r, \mathsf{DE}_1(x))$ into H_1-table.

Recalling the H_1-table in Game 1, we note that, the only case that the system calls $\tilde{H}_1(y, m)$ is when the adversary knows nothing about $\tilde{H}_1(y, m)$, although the adversary might know $\mathsf{DE}(y, m), P(y, m)$ and $\mathsf{NSK}(y, \tilde{H}_1(y, m))$. Therefore, from the adversary's view, $\tilde{H}_1(y, m)$ is uniformly distributed in $\mathcal{X}$, and it is equivalent to randomly pick $r \leftarrow \mathcal{X}$ and implicitly set $r = (y, m, H_1)$, $\mathsf{DE}_1(y, m) = (r, \mathsf{NKG})$ and $P(y, m) \oplus \mathsf{DE}_2(y, m) = (y, r, \mathsf{NSK})$.

Game 4. This game is identical to Game 3, except that to answer NKG queries, where the system only uses the tables and honest interfaces. Assuming (r, NKG) is the k-th query, then

- Case 1. If $Q_k \in \mathsf{NKG}$, then same as above;
- Case 2. If $Q_k \notin \mathsf{NKG}$ but there is a tuple $T = (T_1, T_2, T_3)$ in H_0-table such that $T_2 = r$, then same as above;
- Case 3. If $Q_k \notin \mathsf{NKG}$ but there is a tuple $T = (T_1, T_2, T_3, T_4)$ in H_1-table such that $T_3 = r$, then same as above;
- Case 4. Otherwise, the system randomly samples $x \leftarrow \mathcal{X}$, responds with $\mathsf{DKG}(x)$, and inserts $(x, r, \mathsf{DKG}(x))$ into the table.

Due to definition of the NKG-table, we observe that, the only case that the system calls (r, NKG) is when it knows nothing about x such that $\tilde{H}_0(x) = r$ or (y, m) such that

$\tilde{H}_1(y,m) = r$. As $\tilde{H}_0$ and $\tilde{H}_1$ are random oracles, except with negligible probability, adversary would not ask

Recalling the H_1-table in Game 1, we note that, the only case that the system calls $\tilde{H}_1(y,m)$ is when the adversary knows nothing about $\tilde{H}_1(y,m)$, although the adversary might know $\mathsf{DE}(y,m), P(y,m)$ and $\mathsf{NSK}(y,\tilde{H}_1(y,m))$. Therefore, from the adversary's view, $\tilde{H}_1(y,m)$ is uniformly distributed in $\mathcal{X}$, and it is equivalent to randomly pick $r \leftarrow \mathcal{X}$ and implicitly set $r = (y,m,H_1)$, $\mathsf{DE}_1(y,m) = (r,\mathsf{NKG})$ and $P(y,m) \oplus \mathsf{DE}_2(y,m) = (y,r,\mathsf{NSK})$.

Game 5. The game is identical to Game 4, except that to answer P queries, where the system only uses the tables and honest interfaces. Assuming (y,m,P) is the k-th query, then

- Case 1. If $Q_k \in P \cup P^{-1}$, then it responds with the corresponding output string;
  Note that for the cases below where $Q_k \notin P \cup P^{-1}$, after we compute the response z, we will always add the tuple (y,m,z) to P-table and P^{-1}-table, so that on future identical queries, Q_k will be in $P \cup P^{-1}$.
- Case 2. If $Q_k \notin P \cup P^{-1}$ but there is a tuple $T = (T_1,T_2,T_3,T_4)$ in H_1-table such that $T_4 = \mathsf{DE}_1(y,m)$, then the system searches every tuple $T' = (T'_1,T'_2,T'_3)$ in NSK-tabe such that $T'_1 = y, T'_2 = T_3$. If such a tuple T' is found, then the system responds to the query with $\mathsf{DE}_2(y,m) \oplus T'_3$, else the system checks the following cases.
- Case 3. If $Q_k \notin P \cup P^{-1}$ but there is a tuple $T = (T_1,T_2,T_3)$ in $H_0 \cup \mathsf{NKG}$ such that $T_3 = \mathsf{DE}_1(y,m)$, then the system searches every tuple $T' = (T'_1,T'_2,T'_3)$ in NSK-tabe such that $T'_1 = y, T'_2 = T_2$. If such a tuple T' is found, then the system responds to the query with $\mathsf{DE}_2(y,m) \oplus T'_3$, else the sytem checks the following cases.
- Case 4. If $Q_k \notin P$ but there is tuple $T = (T_1,T_2,T_3)$ in $H_0 \cup \mathsf{NKG}$ such that $T_3 = y$, then the system searches every tuple $T' = (T'_1,T'_2,T'_3)$ in NSK-table such that $T'_1 = \mathsf{DE}_1(y,m), T'_2 = T_2$. If such a tuple is found, then the system responds with $\mathsf{DE}_2(y,m) \oplus T'_3$, else checks the following cases.
- Case 5. If $Q_k \notin P$ but there is tuple $T = (T_1,T_2,T_3,T_4)$ such that $T_4 = y$, then the system searches every tuple $T' = (T'_1,T'_2,T'_3)$ in NSK-table such that $T'_1 = \mathsf{DE}_1(y,m), T'_2 = T_3$. If such a tuple is found, then the system responds with $\mathsf{DE}_2(y,m) \oplus T'_3$.
- Case 6. Otherwise, the system randomly samples $z \leftarrow \mathcal{Z}$ and responds with z.

Game 6. This game is identical to Game 5, except that to answer NSK queries, where the system only uses the tables and honest interfaces. Assuming (y,r,NSK) is the k-th query, then

- Case 1. If $Q_k \in \mathsf{NSK}$, then same as above;
- Case 2. If $Q_k \notin \mathsf{NSK}$ but there is a tuple $(T_1,T_2,T_3) \in \mathsf{NSK}$ such that $T_1 = \mathsf{NKG}(r), y = \mathsf{NKG}(T_2)$, then the system responds with T_3;
- Case 3. If $Q_k \notin \mathsf{NSK}$ but there is tuple $T = (T_1,T_2,T_3,T_4)$ in H_1-table such that $T_1 = y, T_3 = r$, then searches every tuple $T' = (T'_1,T'_2,T'_3) \in P \cup P^{-1}$ such that $T'_1 = y, T'_2 = T_2$. If such a tuple is found, then the system responds with $\mathsf{DE}_2(y,T_2) \oplus T'_3$, else checks the following cases;
- Case 4. If $Q_k \notin \mathsf{NSK}$ but there is a tuple $T = (T_1,T_2,T_3) \in H_0 \cup \mathsf{NKG}$ such that $T_2 = r$, then searches every tuples $T' = (T'_1,T'_2,T'_3) \in P \cup P^{-1}$ such that $\mathsf{DE}_1(T'_1,T'_2) = y$ and $T'_1 = T_3$. If such a tuple is found, then the system repsonds with $\mathsf{DE}_2(T'_1,T'_2) \oplus T'_3$, else checks the following cases.
- Case 5. Otherwise, the system randomly samples $z \leftarrow \mathcal{Z}$ and responds with z.

Game 7. This game is identical to Game 6, except that to answer P^{-1} queries, where the system only uses the tables and honest interfaces. Assuming (z,P^{-1}) is the k-th query, then

- Case 1. If $Q_k \in P \cup P^{-1}$, then same as above;
- Case 2. If $Q_k \in P \cup P^{-1}$ but there are a tuple $T = (T_1, T_2, T_3, T_4) \in H_1$ and $T' = (T'_1, T'_2, T'_3) \in \mathsf{NSK}$ such that $T_1 = T'_1, T_3 = T_2$ and $\mathsf{DE}_2 = z \oplus T'_3$. If such tuples are found, then the system responds with (T_1, T_2), else checks the following cases;
- Case 3. If $Q_k \notin P \cup P^{-1}$ but there are a tuple $T = (T_1, T_2, T_3) \in H_0 \cup \mathsf{NKG}$ and $T' = (T'_1, T'_2, T'_3)$ such that $T_1 \neq *, T_2 = T'_2$ and $\mathsf{DD}(T'_1, z \oplus T'_3, T_1) \neq \perp$, then the system responds with $(T_3, \mathsf{DD}(T'_1, z \oplus T'_3, T_1))$, else checks the following cases;
- Case 4. Otherwise, the system randomly picks $y \leftarrow \mathcal{Y}, m \leftarrow \mathcal{M}$, and responds with (y, m).

Game 8. In Game 7, the queries to $H_0, H_1, P, P^{-1}\mathsf{NKG}, \mathsf{NSK}$ are answered by the tables which're maintained by the system and by making queries to $\mathsf{DKG}, \mathsf{DE}, \mathsf{DD}$. The system never makes queries directly to $\tilde{H}_0, \tilde{H}_1, \tilde{P}, \tilde{P}^{-1}, \tilde{\mathsf{NKG}}, \tilde{\mathsf{NSK}}$; these oracles are *only* used to answer the $\mathsf{DKG}, \mathsf{DE}, \mathsf{DD}$ queries (either generated by the adversary or by the system's response to $H_0, H_1, P, P^{-1}, \mathsf{NKG}, \mathsf{NSK}$ queries). At this point, it is straightforward to show that we can replace $\mathsf{DKG}, \mathsf{DE}$ and DD with the ideal versions from Definition 6, resulting in Game 8.

We note that in Game 8, the system is efficient, and it responds to the adversarial interfaces just by keeping several tables and accessing the honest interfaces. Thus, we can build a simulator that responds to $H_0, H_1, P, P^{-1}, \mathsf{NKG}, \mathsf{NSK}$ queries exactly as the system does in Game 8. The result is that Game 8 corresponds to the ideal world. Since Game 0 is the real world, it suffices to prove that any adjacent games are indistinguishable. Next we will give the description of our simulator and prove the indistinguishability between each adjacent games.

Simulator. Let $\Pi_{\mathsf{DPKE}} = (\mathsf{DKG}, \mathsf{DE}, \mathsf{DD})$ be an ideal DPKE, associated with secret key space $\mathcal{X}$, public key space $\mathcal{Y}$, message space $\mathcal{M}$ and ciphertext space $\mathcal{C}$, where 1)$|\mathcal{X}| \leq |\mathcal{Y}|$; 2) $\mathcal{C} = \mathcal{Y} \times \mathcal{Z}$; 3)$\mathcal{Z} = \mathcal{Y} \times \mathcal{M}$. The simulator $\mathcal{S}$ responds to the queries as follows:

H_0-QUERY. Suppose (x, H_0) is the k-th query, the system responds as follows:

- Case 1. If $Q_k \in H_0$, then it responds with the corresponding output string;
- Case 2. If $Q_k \notin H_0$ but there is a tuple $T = (T_1, T_2, T_3)$ in NKG-table such that $T_3 = \mathsf{DKG}(x)$, then responds with T_2 and inserts (x, T_2, T_3) into H_0 table;
- Case 3. If $Q_k \notin H_0$ but there is a tuple $T = (T_1, T_2, T_3, T_4)$ in H_1-table such that $T_4 = \mathsf{DKG}(x)$, then responds with T_3 and inserts (x, T_3, T_4) into H_0 table;
- Case 4. Otherwise, the systems randomly samples $r \leftarrow \mathcal{X}$, responds with r and inserts $(x, r, \mathsf{DKG}(x))$ into H_0-table.

H_1-QUERY.Suppose (y, m, H_1) is the k-th query, the system responds as follows:

- Case 1. If $Q_k \in H_1$, then it responds with the corresponding output string;
- Case 2. If $Q_k \notin H_1$ but there is a tuple $T = (T_1, T_2, T_3)$ in NKG-table such that $T_3 = \mathsf{DE}_1(y, m)$, then responds with T_2 and inserts (y, m, T_2, T_3) into H_1 table;
- Case 3. If $Q_k \notin H_1$ but there is a tuple $T = (T_1, T_2, T_3)$ in H_0-table such that $T_3 = \mathsf{DE}_1(y, m)$, then responds with T_2 and inserts (y, m, T_2, T_3) into H_1 table;
- Case 4. Otherwise, the systems randomly samples $r \leftarrow \mathcal{X}$, responds with r and inserts $(y, m, r, \mathsf{DE}_1(x))$ into H_1-table.

NKG-QUERY. Suppose (r, NKG) is the k-th query, the system responds as follows:

- Case 1. If $Q_k \in \mathsf{NKG}$, then it responds with the corresponding output string;
- Case 2. If $Q_k \notin \mathsf{NKG}$ but there is a tuple $T = (T_1, T_2, T_3)$ in H_0-table such that $T_2 = r$, then responds with T_3 and inserts (T_1, T_2, T_3) into NKG table;

- Case 3. If $Q_k \notin \mathsf{NKG}$ but there is a tuple $T = (T_1, T_2, T_3, T_4)$ in H_1-table such that $T_3 = r$, then responds with T_4 and inserts $(*, T_3, T_4)$ into NKG table;
- Case 4. Otherwise, the system randomly samples $x \leftarrow \mathcal{X}$, responds with $\mathsf{DKG}(x)$, and inserts $(x, r, \mathsf{DKG}(x))$ into the table.

P-QUERY. Suppose (y, m, P) is the k-th query, the system responds as follows:

- Case 1. If $Q_k \in P \cup P^{-1}$, then it responds with the corresponding output string;
  Note that for the cases below where $Q_k \notin P \cup P^{-1}$, after we compute the response z, we will always add the tuple (y, m, z) to P-table and P^{-1}-table, so that on future identical queries, Q_k will be in $P \cup P^{-1}$.
- Case 2. If $Q_k \notin P \cup P^{-1}$ but there is a tuple $T = (T_1, T_2, T_3, T_4)$ in H_1-table such that $T_4 = \mathsf{DE}_1(y, m)$, then the system searches every tuple $T' = (T'_1, T'_2, T'_3)$ in NSK-tabe such that $T'_1 = y, T'_2 = T_3$. If such a tuple T' is found, then the system responds to the query with $\mathsf{DE}_2(y, m) \oplus T'_3$, else the system checks the following cases.
- Case 3. If $Q_k \notin P \cup P^{-1}$ but there is a tuple $T = (T_1, T_2, T_3)$ in $H_0 \cup \mathsf{NKG}$ such that $T_3 = \mathsf{DE}_1(y, m)$, then the system searches every tuple $T' = (T'_1, T'_2, T'_3)$ in NSK-tabe such that $T'_1 = y, T'_2 = T_2$. If such a tuple T' is found, then the system responds to the query with $\mathsf{DE}_2(y, m) \oplus T'_3$, else the sytem checks the following cases.
- Case 4. If $Q_k \notin P$ but there is tuple $T = (T_1, T_2, T_3)$ in $H_0 \cup \mathsf{NKG}$ such that $T_3 = y$, then the system searches every tuple $T' = (T'_1, T'_2, T'_3)$ in NSK-table such that $T'_1 = \mathsf{DE}_1(y, m), T'_2 = T_2$. If such a tuple is found, then the system responds with $\mathsf{DE}_2(y, m) \oplus T'_3$, else checks the following cases.
- Case 5. If $Q_k \notin P$ but there is tuple $T = (T_1, T_2, T_3, T_4)$ such that $T_4 = y$, then the system searches every tuple $T' = (T'_1, T'_2, T'_3)$ in NSK-table such that $T'_1 = \mathsf{DE}_1(y, m), T'_2 = T_3$. If such a tuple is found, then the system responds with $\mathsf{DE}_2(y, m) \oplus T'_3$.
- Case 6. Otherwise, the system randomly samples $z \leftarrow \mathcal{Z}$ and responds with z.

NSK-QUERY. Suppose (y, r, NSK) is the k-th query, the system responds as follows:

- Case 1. If $Q_k \in \mathsf{NSK}$, then it responds with the corresponding output string;
- Case 2. If $Q_k \notin \mathsf{NSK}$ but there is tuple $T = (T_1, T_2, T_3, T_4)$ in H_1-table such that $T_1 = y, T_3 = r$, then searches every tuple $T' = (T'_1, T'_2, T'_3) \in P \cup P^{-1}$ such that $T'_1 = y, T'_2 = T_2$. If such a tuple is found, then the system responds with $\mathsf{DE}_2(y, T_2) \oplus T'_3$, else checks the following cases;
- Case 3. If $Q_k \notin \mathsf{NSK}$ but there is a tuple $T = (T_1, T_2, T_3) \in H_0 \cup \mathsf{NKG}$ such that $T_2 = r$, then searches every tuples $T' = (T'_1, T'_2, T'_3) \in P \cup P^{-1}$ such that $\mathsf{DE}_1(T'_1, T'_2) = y$ and $T'_1 = T_3$. If such a tuple is found, then the system repsonds with $\mathsf{DE}_2(T'_1, T'_2) \oplus T'_3$, else checks the following cases.
- Case 4. OOtherwise, the system randomly samples $z \leftarrow \mathcal{Z}$ and responds with z.

P^{-1}-QUERY. Suppose (z, P^{-1}) is the k-th query, the system responds as follows:

- Case 1. If $Q_k \in P \cup P^{-1}$, then it responds with the corresponding output string;
- Case 2. If $Q_k \in P \cup P^{-1}$ but there are a tuple $T = (T_1, T_2, T_3, T_4) \in H_1$ and $T' = (T'_1, T'_2, T'_3) \in \mathsf{NSK}$ such that $T_1 = T'_1, T_3 = T_2$ and $\mathsf{DE}_2 = z \oplus T'_3$. If such tuples are found, then the system responds with (T_1, T_2), else checks the following cases;
- Case 3. If $Q_k \notin P \cup P^{-1}$ but there are a tuple $T = (T_1, T_2, T_3) \in H_0 \cup \mathsf{NKG}$ and $T' = (T'_1, T'_2, T'_3)$ such that $T_1 \neq *, T_2 = T'_2$ and $\mathsf{DD}(T'_1, z \oplus T'_3, T_1) \neq \perp$, then the system responds with $(T_3, \mathsf{DD}(T'_1, z \oplus T'_3, T_1))$, else checks the following cases;

– Case 4. Otherwise, the system randomly picks $y \leftarrow \mathcal{Y}, m \leftarrow \mathcal{M}$, and responds with (y, m).

Note that $\mathcal{S}$ works exactly the same as the system in Game 8. Moreover, the distribution of the honest interfaces in Game 8 is also identical to the ones in the ideal game, which means, for any adversary, the advantage in both Game 8 and Ideal Game is identical. Therefore, it suffices to prove that any adjacent games are indistinguishable.

Claim. Game 0 $\approx$ Game 1.

Proof. It's trivial to note that Pr[Game 0] = Pr[Game 1], as in Game 1, the system only changes the way of maintaining the tables and the terms stored in each tuple is identical to the responses of real oracles.

Claim. Game 1 $\approx$ Game 2.

Proof. Firstly, we note that, in either Game 1 or Game 2, the adversary's view on the honest interfaces is consistent, hence it suffices to prove that, with high probability, the view on the adversarial interfaces also does not change. According to the definition, the only difference between Game 1 and Game 2 is when Case 4 occurs. Suppose (x, H_0) is the k-th query, the adversary's view on the adversarial interfaces are as follows:

View on H_0. By definition, $\tilde{H_0}$ is a random oracle such that 1) the response is consistent; 2) the response to a fresh input is uniformly random.

Thus, the adversary's view on H_0 is consistent in both games, if the responses of H_0 queries satisfy the two properties above. When case 4 occurs, after Q_k, the system sets $H_0(\mathsf{x}) = \mathsf{r}$. As (x, H_0) never appears in the previous queries, property 1 and 2 preserve.

View on H_1. By definition, $\tilde{H_1}$ is a random oracle such that 1) the response is consistent; 2) the response to a fresh input is uniformly random.

Thus, the adversary's view on H_1 is consistent in both games, if the responses of H_1 queries satisfy the two properties above. We note that unless adversary can output X, Y, M such that $\tilde{H_0}(X) = \tilde{H_1}(Y, M)$, the response of H_1 queries are independent of H_0-table.

View on NKG. By definition, $\tilde{\mathsf{NKG}}$ is the key generation algorithm of an ideal NIKE such that 1) the response is consistent; 2)the response to a fresh input is uniformly random; 3) the response is injective; 4) $\mathsf{NKG}(H_0(\mathsf{x})) = \mathsf{DKG}(\mathsf{x})$; 5) $\mathsf{NKG}(H_1(y, m)) = \mathsf{DE}_1(\mathsf{y}, \mathsf{m})$. Thus, the adversary's view on NKG is consistent in both games, if the responses of NKG queries satisfy the five properties above. Property 4 and 5 hold trivially and for other properties, we apply the same technique in the analysis of Theorem 5. We note that if r never appears in the previous queries, and $\tilde{H_0}(x)$ and x^* such that $\tilde{H_0}(\mathsf{x}^*) = \mathsf{r}$ never appear during the queries, then property 1, 2 and 3 hold. This bad event can be bounded by $\frac{3q}{|\mathcal{X}|}$.

View on P. By definition, $\tilde{P}$ is a random permutation such that 1) the response is consistent; 2) the response to a fresh input is uniformly random; 3) the response is injective.

Thus, the adversary's view on P is consistent in both games, if the responses of P queries satisfy the three properties above. In fact, in both games, the P queries are answered using NSK-table and honest interfaces, hence it suffices to prove the view on NSK does not change with high probability.

View on P^{-1}. By definition, $\tilde{P^{-1}}$ is a random permutation such that 1) the response is consistent; 2) the response to a fresh input is uniformly random; 3) the response is injective; 4) P^{-1} is P's inverse.

Thus, the adversary's view on P^{-1} is consistent in both games, if the responses of P^{-1} queries satisfy the three properties above. Same as the case on P, we note that the P^{-1} queries are also answered using NSK-table and honest interfaces.

View on NSK. By definition, $\tilde{\mathsf{NSK}}$ is the shared-key algorithm of an ideal NIKE such that 1) the response is consistent; 2)the response to a fresh input is uniformly random; and for any $\mathsf{x}, \mathsf{x}_1, \mathsf{x}_2, \mathsf{T}, \mathsf{T}_1, \mathsf{T}_2 \in \mathcal{X}, \mathsf{y}, \mathsf{y}_1, \mathsf{y}_2 \in \mathcal{Y}, \mathsf{m}, \mathsf{m}_1, \mathsf{m}_2 \in \mathcal{M}$, we have

3)$\mathsf{NSK}(\mathsf{NKG}(\mathsf{T}_1), \mathsf{T}_2) = \mathsf{NSK}(\mathsf{NKG}(\mathsf{T}_2), \mathsf{T}_1)$;

4)$\mathsf{NSK}(\mathsf{NKG}(\mathsf{x}), \mathsf{T}) = \mathsf{NSK}(\mathsf{NKG}(\mathsf{T}), H_0(\mathsf{x}))$;

5)$\mathsf{NSK}(\mathsf{DE}_1(\mathsf{y}, \mathsf{m}), \mathsf{T}) = \mathsf{NSK}(\mathsf{NKG}(\mathsf{T}), H_1(\mathsf{y}, \mathsf{m}))$;

6)$\mathsf{NSK}(\mathsf{DKG}(\mathsf{x}_1), H_0(\mathsf{x}_2)) = \mathsf{NSK}(\mathsf{DKG}(\mathsf{x}_2), H_0(\mathsf{x}_1))$;

7)$\mathsf{NSK}(\mathsf{DKG}(\mathsf{x}), H_1(\mathsf{y}, \mathsf{m})) = \mathsf{NSK}(\mathsf{DE}_1(\mathsf{y}, \mathsf{m}), H_0(\mathsf{x}))$;

8)$\mathsf{NSK}(\mathsf{DE}_1(\mathsf{y}_1, \mathsf{m}_1), H_1(\mathsf{y}_2, \mathsf{m}_2)) = \mathsf{NSK}(\mathsf{DE}_1(\mathsf{y}_2, \mathsf{m}_2), H_1(\mathsf{y}_1, \mathsf{m}_1))$;

9)$\mathsf{NSK}(\mathsf{y}, H_1(\mathsf{y}, \mathsf{m})) \oplus P(\mathsf{y}, \mathsf{m}) = \mathsf{DE}_2(\mathsf{y}, \mathsf{m})$;

10)$\mathsf{NSK}(\mathsf{DE}_1(\mathsf{DKG}(\mathsf{x}), \mathsf{m}), H_0(\mathsf{x})) \oplus P(\mathsf{DKG}(\mathsf{x}), \mathsf{m}) = \mathsf{DE}_2(\mathsf{DKG}(\mathsf{x}), \mathsf{m})$;

11)$\mathsf{NSK}(\mathsf{DE}_1(\mathsf{NKG}(\mathsf{T}), \mathsf{m}), \mathsf{T}) \oplus P(\mathsf{NKG}(\mathsf{T}), \mathsf{m}) = \mathsf{DE}_2(\mathsf{NKG}(\mathsf{T}), \mathsf{m})$.

Thus, the adversary's view on NSK is consistent in both games, if the responses of NSK queries satisfy the eleven properties, and those equations above are all possible consistency checks during the indifferentiability proof. It's trivial to note that, in both games those equations always hold. And for property 1 and 2, we still apply the technique in the analysis of Theorem 5. If $\mathsf{NSK}(*, \mathsf{r})$ never appears in the previous queries, and $\mathsf{NSK}(*, H_0(\mathsf{x}))$ and $\mathsf{NSK}(*, H_0(\mathsf{x}^*))$ such that $\tilde{H}_0(\mathsf{x}^*) = \mathsf{r}$ never appears, then property 1 and 2 hold. And we could bound this bad event by $\frac{3q|\mathcal{Y}|}{|\mathcal{Z}|} = \frac{3q}{|\mathcal{M}|}$.

Now we bound the union of those bad events as:

$$\Pr[\mathsf{Bad}] \leq \frac{q}{|\mathcal{X}|} + \frac{3q}{|\mathcal{X}|} + \frac{3q}{|\mathcal{M}|} \leq \mathsf{negl}(\lambda),$$

which refers to

$$|\Pr[\text{Game } 1] - \Pr[\text{Game } 2]| \leq q \cdot \Pr[\mathsf{Bad}] \leq \mathsf{negl}(\lambda).$$

Claim. Game 2 $\approx$ Game 3.

Proof. Easy to note that the analysis in this Claim is symmetrical to the last one, hence

$$|\Pr[\text{Game } 2] - \Pr[\text{Game } 3]| \leq |\Pr[\text{Game } 1] - \Pr[\text{Game } 2]|.$$

Claim. Game 3 $\approx$ Game 4.

Proof. Firstly, we note that, in either Game 1 or Game 2, the adversary's view on the honest interfaces is consistent, hence it suffices to prove that, with high probability, the view on the adversarial interfaces also does not change. According to the definition, the only difference between Game 3 and Game 4 is when Case 4 occurs. Suppose $(\mathsf{T}, \mathsf{NKG})$ is the k-th query, in Game 4 the system randomly samples $\mathsf{x} \leftarrow \mathcal{X}$ and responds with $\mathsf{DKG}(\mathsf{x})$. Now, let's see the adversary's view on the adversarial interfaces.

View on H_0 and H_1. Easy to note that the responses of H_0 and H_1 queries are consistent except x or x^* such that $\tilde{H_0}(\mathsf{x}^*) = \mathsf{y}$ or $(\mathsf{PK}, \mathsf{M})$ such that $\tilde{H_1}(\mathsf{PK}, \mathsf{M}) = \mathsf{y}$ appear and this bad event is bounded by $\frac{3q}{|\mathcal{X}|}$.

View on NKG. By definition, $\tilde{\mathsf{NKG}}$ is the key generation algorithm of an ideal NIKE such that 1) the response is consistent; 2)the response to a fresh input is uniformly random; 3) the response is injective; 4) $\mathsf{NKG}(H_0(\mathsf{x})) = \mathsf{DKG}(\mathsf{x})$; 5) $\mathsf{NKG}(H_1(y, m)) = \mathsf{DE_1}(\mathsf{y}, \mathsf{m})$.

Easy to note that, property 4 and 5 hold trivially. And if $\mathsf{DKG}(\mathsf{x})$ and x never appears in the previous queries, then property 1 and 2 also hold. For property 3, if no x^* such that $\tilde{H_0}(\mathsf{x}^*) = \mathsf{y}$ or $(\mathsf{PK}, \mathsf{M})$ such that $\tilde{H_1}(\mathsf{PK}, \mathsf{M}) = \mathsf{y}$ appear, then except with collision, property 3 holds. We can bound it by $\frac{5q}{|\mathcal{X}|}$.

View on P and P^{-1}. Same as above.

View on NSK. By definition, $\tilde{\mathsf{NSK}}$ is the shared-key algorithm of an ideal NIKE such that 1) the response is consistent; 2)the response to a fresh input is uniformly random and nine equations above.

Easy to note that those equations hold, and for property 1 and 2, we still apply the techinique in Theorem 5, and have the following: if $\mathsf{NSK}(\mathsf{DKG}(\mathsf{x}), *)$ never appears in the previous queries, and $\mathsf{NSK}(*, \tilde{H_0}(\mathsf{x}))$ and $\mathsf{NSK}(*, \tilde{H_0}(\mathsf{x}^*))$ never appears, then property 1 and 2 hold. For this bad event, we bound it by $\frac{q|\mathcal{M}|}{|\mathcal{Z}|} + \frac{2q|\mathcal{Y}|}{|\mathcal{Z}|} = \frac{q}{|\mathcal{Y}|} + \frac{2q}{|\mathcal{M}|}$.

Now, we bound the union as:

$$\Pr[\mathsf{Bad}] \leq \frac{3q}{|\mathcal{X}|} + \frac{5q}{|\mathcal{X}|} + \frac{q}{|\mathcal{Y}|} + \frac{2q}{|\mathcal{M}|} \leq \mathsf{negl}(\lambda),$$

which refers to

$$|\Pr[\text{Game 3}] - \Pr[\text{Game 4}]| \leq q \cdot \Pr[\mathsf{Bad}] \leq \mathsf{negl}(\lambda).$$

Claim. Game 4 $\approx$ Game 5.

Proof. Firstly, we note that, in either Game 4 or Game 5, the adversary's view on the honest interfaces is consistent, hence it suffices to prove that, with high probability, the view on the adversarial interfaces also does not change. According to the definition, the only difference between Game 4 and Game 5 is when Case 6 occurs where the system replaces the response of P queries (fails in the consistency check) with a random string. Suppose $(\mathsf{y}, \mathsf{m}, P)$ is the k-th query, the adversary's view on the adversarial interfaces are as follows:

View on H_0, H_1, NKG. Note that in both games, the system responds H_0, H_1, NKG queries in the same way, hence the view is consistent.

View on P. When case 6 happens, the system implictly sets $P(\mathsf{y}, \mathsf{m}) = \mathsf{z}$, where $z \leftarrow \mathcal{Z}$. Hence if z never appears in the previous queries, then the view on P preserves.

View on NSK. When case 6 happens, the system implicitly sets

$$\mathsf{NSK}(\mathsf{y}, H_1(\mathsf{y}, \mathsf{m})) = \mathsf{z} \oplus \mathsf{DE_2}(\mathsf{y}, \mathsf{m})$$

Hence, similar to the analysis above, if $\mathsf{z} \oplus \mathsf{DE_2}(\mathsf{y}, \mathsf{m})$ never appears in the previous queries, and $\tilde{P}(\mathsf{y}, \mathsf{m})$ and $(\mathsf{y}', \mathsf{r}')$ such that $\tilde{\mathsf{NSK}}(y', r') = \mathsf{z} \oplus \mathsf{DE_2}(\mathsf{y}, \mathsf{m})$ never appear, then the view on NSK does not change. For the bad event we can bound it by $\frac{3q}{|\mathcal{Z}|}$.

View on P^{-1}. When case 6 occurs, the system implicitly sets $\tilde{P^{-1}}(\mathsf{z} \oplus \mathsf{DE_2}(\mathsf{y}, \mathsf{m})) = (\mathsf{y}, \mathsf{m})$. Hence, similar to the analysis above, if $\mathsf{z} \oplus \mathsf{DE_2}(\mathsf{y}, \mathsf{m})$ never appears in the previous queries, and $\tilde{P}(\mathsf{y},\mathsf{m})$ and z' such that $\tilde{P^{-1}}(\mathsf{z}') = (\mathsf{y},\mathsf{m})$ never appears, then the view on P^{-1} preserves. For this bad event, we can also bound it by $\frac{3q}{|\mathcal{Z}|}$.

Now, we bound the union of these bad events as

$$\Pr[\mathsf{Bad}] \leq \frac{q}{|\mathcal{Z}|} + \frac{3q}{|\mathcal{Z}|} + \frac{3q}{|\mathcal{Z}|} \leq \mathsf{negl}(\lambda),$$

which refers to

$$|\Pr[\text{Game } 4] - \Pr[\text{Game } 5]| \leq q\Pr[\mathsf{Bad}] \leq \mathsf{negl}(\lambda).$$

Claim. Game 5 $\approx$ Game 6.

Proof. Firstly, we note that, in either Game 5 or Game 6, the adversary's view on the honest interfaces is consistent, hence it suffices to prove that, with high probability, the view on the adversarial interfaces also does not change. According to the definition, the only difference between Game 5 and Game 6 is when Case 4 occurs where the system replaces the response of NSK queries (fails in the consistency check) with a random string. Suppose $(\mathsf{y},\mathsf{r},\mathsf{NSK})$ is the k-th query, the adversary's view on the adversarial interfaces are as follows:

View on H_0, H_1, NKG. Same as above.

View on P. In both games, the responses of P queries are responded by NSK table and P^{-1} table or a random string, therefore expect with collision, the view on P does not change. And this bad event can be bounded by $\frac{q}{|\mathcal{Z}|}$.

View on NSK. When case 4 occurs, then the system sets $\mathsf{NSK}(\mathsf{y},\mathsf{r}) = \mathsf{z}$ where $\mathsf{z} \leftarrow \mathcal{Z}$. As z is randomly sampled, the view on NSK in both games is consistent.

View on P^{-1}. When case 4 occurs, if there exists (PK,M) such that $\mathsf{NSK}(\mathsf{PK},\mathsf{M}) = \mathsf{NSK}(\mathsf{y},\mathsf{r})$, then the system sets $P^{-1}(\mathsf{z} \oplus \mathsf{DE}_2(\mathsf{PK},\mathsf{M})) = (\mathsf{PK},\mathsf{M})$. Hence similar to the analysis above, if $\mathsf{z} \oplus \mathsf{DE}_2(\mathsf{PK},\mathsf{M})$ never appears in the previous queries, and $\tilde{\mathsf{NSK}}(\mathsf{y},\mathsf{r})$ and $\tilde{P}(\mathsf{PK},\mathsf{M})$ never appear, then the view on P^{-1} do not change. And we can bound this bad event by $\frac{3q}{|\mathcal{Z}|}$.

Now, we bound the union of these bad events as:

$$\Pr[\mathsf{Bad}] \leq \frac{q}{|\mathcal{Z}|} + \frac{3q}{|\mathcal{Z}|} \leq \mathsf{negl}(\lambda),$$

which refers to

$$|\Pr[\text{Game } 5] - \Pr[\text{Game } 6]| \leq q\Pr[\mathsf{Bad}] \leq \mathsf{negl}(\lambda).$$

Claim. Game 6 $\approx$ Game 7.

Proof. Firstly, we note that, in either Game 6 or Game 7, the adversary's view on the honest interfaces is consistent, hence it suffices to prove that, with high probability, the view on the adversarial interfaces also does not change. According to the definition, the only difference between Game 6 and Game 7 is when Case 4 occurs where the system replaces the response of P^{-1} queries (fails in the consistency check) with a random string. Suppose $(\mathsf{z},\mathsf{P}^{-1})$ is the k-th query, the adversary's view on the adversarial interfaces are as follows:

View on H_0, H_1, NKG. Same as above.

View on P and NSK. In both games, the responses of P queries are responded by P^{-1} table or a random string, therefore expect with collision, the view on P and NSK does not change. And this bad event can be bounded by $\frac{2q}{|\mathcal{Z}|}$.

View on P^{-1}. When case 4 occurs(the consistency check fails), then the system replaces $P^{-1}(z)$ with a random string.

Firstly we note that, in both games, the real oracles $\tilde{H}_0, \tilde{H}_1, \tilde{\mathsf{NKG}}, \tilde{NSK}, \tilde{P}$ are hidden, hence except with negligible probability($\leq \frac{q}{|\mathcal{Z}|}$), the adversary can not output a tuple $(\mathsf{PK}, \mathsf{M}, \mathsf{Z})$, such that $\tilde{P}(\mathsf{PK}, \mathsf{M}) = \mathsf{Z}$. Hence, it's rest to show if the consistency check fails, with high probability, the view on P^{-1} does not change. In fact, the view changes means that the adversary, without knowing $\mathsf{NSK}(\mathsf{y}, \mathsf{m})$, can output a z such that $z = \mathsf{NSK}(\mathsf{y}, H_1(\mathsf{y}, \mathsf{m})) \oplus \mathsf{DE}_2(\mathsf{y}, \mathsf{m})$.

$$|\Pr[\text{Game } 6] - \Pr[\text{Game } 7]| \leq q \cdot \frac{2q}{|\mathcal{Z}|} \leq \mathsf{negl}(\lambda).$$

Claim. Game 7 $\approx$ Game 8.

Proof. Due to definition, we note that in Game 7, the system responds all of the adversarial interfaces just using tables and honest interfaces, it never directly calls the real oracles. Besides, $\tilde{H}_0, \tilde{H}_1$ are random oracles, $(\mathsf{NKG}, \mathsf{NSK})$ is an ideal NIKE and $\tilde{P}$ is random permutations, hence unless the adversary can output $\mathsf{x} \neq \mathsf{x}' \in \mathcal{X}$ such that $\tilde{H}_0(\mathsf{x}) = \tilde{H}_0(\mathsf{x}')$ or $(y, m) \neq (y', m') \in \mathcal{Y} \times \mathcal{M}$ such that $\tilde{H}_1(y, m) = \tilde{H}_1(y', m')$ or $x, (y, m)$ such that $\tilde{H}_0(\mathsf{x}) = \tilde{H}_1(y, m)$, the view on Game 7 or Game 8 is identical. Therefore,

$$|\Pr[\text{Game } 7] - \Pr[\text{Game } 8]| \leq \frac{q^2}{|\mathcal{X}|} \leq \mathsf{negl}(\lambda).$$

Combining together, we complete the whole proof. □

9 Proof of Theorem 8

In this section, we give the full proof of Theorem 8, that we show Π_{PKE} is indifferentiable from an ideal PKE.

Proof. According to the definition of indifferentiability, we note that, in the real game, the adversary has three honest interfaces $(\mathsf{PKG}, \mathsf{PE}, \mathsf{PD})$ and four adversarial interfaces $(H_0, \mathsf{DKG}, \mathsf{DE}, \mathsf{DD})$. Therefore, we need to build an efficient simulator that can simulate these adversarial interfaces properly, which means, for any PPT differentiator $\mathcal{D}$, the view of $\mathcal{D}$ in the real game is computationally close to the view in the ideal game. To do so, we will go through with a sequence of hybrid games. Before the description of the games, we first specify some parameters:

- there are seven types of queries such as $(y, m, r, H_0), (x, \mathsf{DKG}), (y, m, r, \mathsf{DE}), (x, c, \mathsf{DD}), (x, \mathsf{PKG}), (y, m, r, \mathsf{PE}), (x, c, \mathsf{PD})$, where $x \leftarrow \mathcal{X}, y \leftarrow \mathcal{Y}, r \leftarrow \mathcal{R}$ and $c \leftarrow \mathcal{C}$;
- adversary makes q queries to the system, where $q = \mathsf{poly}(\lambda)$;
- the real oracles used in the construction are $\tilde{H}_0, \tilde{\mathsf{DKG}}, \tilde{\mathsf{DE}}, \tilde{\mathsf{DD}}$.

Now we are ready the define the games.

Game 0. This game is identical to the real game, where every query is answered by real oracles. For instance, the response of (x, H_0) is $\tilde{H}_0(x)$, the response of (x, PKG) is $\tilde{\mathsf{DKG}}(x)$ and so forth. Moreover, during the queries, it also maintains three tables, referring to H_0-table, DE-table and DD-table. Specifically,

- H_0-table: Initially empty, consist of tuples with form of (y, m, r, str). Once the adversary makes a (y, m, r, H_0) query which does not exists in H_0 table (no tuple that the first element of it is r), it inserts $(y, m, r, \tilde{H}_0(y, m, r))$ into the table.
- DE-table: Initially empty, consists of tuples with form of (y, m, r, str). Once the adversary makes a (y, m, r, DE) query which does not exist in DE-table, it inserts $(y, m, r, \tilde{\mathsf{DE}}(y, m, r))$ into the table.
- DD-table: Initially empty, consists of tuples with form of $(x, c, \mathsf{str}_1, \mathsf{str}_2)$(the first string is plaintext and the second one is the random nonce). Once the adversary makes a (x, c, DD) query, which does not exist in DD-table, it inserts $(x, c, \tilde{\mathsf{DD}}(x, c))$ into the table.

Note that at this point these tables are just keeping track of information relating to adversary's queries, and are completely hidden to the adversary. Now, we define a relation based on the tables. We say a H_0 query $Q_k = (y, m, r, H_0)$ is in a table K, denoted $Q_k \in K$, if there is a 4-tuple $T = (T_1, T_2, T_3, T_4) \in K$ such that $T_1 = y, T_2 = m, T_3 = r$. Analogously, we say a DE query $(y, m, r, \mathsf{DE}) \in \mathsf{K}$ if there is $T = (T_1, T_2, T_3, T_4) \in K$ such that $T_1 = y, T_2 = m, T_3 = r$; and we say a DD query $(x, c, \mathsf{DD}) \in \mathsf{K}$ if there is a tuple $(T_1, T_2, T_3, T_4) \in K$ such that $T_1 = x, T_2 = c$.

Then, we show a game that the system responds to the queries by using both tables and the real oracles without changing adversary's view.

Game 1. This game is identical to Game 0, except the way of maintaining the tables and responding to the queries. Specifically,

H_0-QUERY. Suppose (y, m, r, H_0) is the k-th query, the system responds as follows:

- Case 1. If $Q_k \in H_0$, then it responds with the corresponding output string;
  Note that for cases below where $Q_k \notin H_0$, after we computes str, we will always add the tuple (y, m, r, str) to H_0, so that on future identical queries, Q_k will be in H_0.
- Case 2. If $Q_k \notin H_0$ but there is a tuple $(T_1, T_2, T_3, T_4) \in \mathsf{DE}$-table such that $T_1 = y, T_2 = m, T_3 = \mathsf{str}||0^{n_3}$ and $T_4 = \mathsf{PE}(y, m, r)$, then the system responds with str, else checks the following cases;
- Case 3. If $Q_k \notin H_0$ but there is a tuple $(T_1, T_2, T_3, T_4) \in \mathsf{DD}$-table such that $T_3 = m, \mathsf{PKG}(T_1) = y, \mathsf{PE}(y, m, r) = T_2$ and $T_4 = \mathsf{str}||0^{n_3}$, then the system reponds with str, else checks the following cases;
- Case 4. Otherwise, the system responds with $\tilde{H}_0(y, m, r)$ and inserts $(y, m, r, \tilde{H}_0(y, m, r))$ and $(y, m, \tilde{H}_0(y, m, r)||0^{n_3}, \mathsf{PE}(y, m, r))$ into H_0-table and DE-table, respectively.

DKG-QUERY. Suppose (x, DKG) is the k-th query, the system responds with $\mathsf{PKG}(x)$.

DE-QUERY. Suppose (y, m, r, DE) is the k-th query(for ease, we split $r = r_1||r_2$), the system responds as follows:

- Case 1. If $Q_k \in \mathsf{DE}$, then it responds with the corresponding output string;
- Case 2. If $Q_k \notin \mathsf{DE}$ and $r_2 \neq 0^{n_3}$, then
  1. If there is a tuple $T = (T_1, T_2, T_3, T_4)$ in DD-table such that $y = \mathsf{PKG}(T_1)$, $T_3 = m$ and $T_4 = r$, then the system responds with T_2;
  2. Otherwise the system responds the query with $\tilde{\mathsf{DE}}(y, m, r)$.
- Case 3. If $Q_k \notin \mathsf{DE}$ and $r_2 = 0^{n_3}$, then
  1. If there is a tuple $T = (T_1, T_2, T_3, T_4)$ in DD-table such that $y = \mathsf{PKG}(T_1)$, $T_3 = m$ and $T_4 = r$, then the system responds with T_2;
  2. If there is $(T_1, T_2, T_3, T_4) \in H_0$ such that $T_1 = y, T_2 = m, T_4 = r||0^{n_3}$, then the system responds with $\mathsf{PE}(y, m, T_3)$;

3. Otherwise the system responds the query with $\tilde{\mathsf{DE}}(y, m, r)$.

DD-QUERY. Suppose (x, c, DD) is the k-th query, the system responds as follows:

- Case 1. If $Q_k \in \mathsf{DD}$, then it responds with the corresponding output string;
  Note that for cases below where $Q_k \notin \mathsf{DD}$, after we computes the value $\mathsf{str}_1, \mathsf{str}_2$, we will always add the tuple $(x, c, \mathsf{str}_1, \mathsf{str}_2)$ into DD, so that on future identical queries, $Q_k \in \mathsf{DD}$.
- Case 2. If $Q_k \notin \mathsf{DD}$ but there is a tuple $(T_1, T_2, T_3, T_4) \in \mathsf{DE}$ such that $T_1 = \mathsf{PKG}(x), T_4 = c$, then the system responds with (T_2, T_3);
- Case 3. Otherwise, the system calls $m^* = (x, c, \mathsf{PD})$, and responds as follows:
  1. If $m^* =\perp$, then the system aborts.
  2. Else if there is a tuple $(T_1, T_2, T_3, T_4) \in H_0$ such that $T_1 = \mathsf{PKG}(x) = T_1, T_2 = m^*$ and $\mathsf{PE}(T_1, T_2, T_3) = c$, then the system responds with $(\mathsf{str}_1, \mathsf{str}_2) = (T_2, T_4||0^{n_3})$;
  3. Otherwise, the system responds with $\tilde{\mathsf{DD}}(x, c)$.

Game 2. This game is identical to Game 1, except that to answer H_0 queries, where the system only uses the tables and honest interfaces. Assuming (y, m, r, H_0) is the k-th query, then

- Case 1. If $Q_k \in H_0$, then it responds with the corresponding output string;
- Case 2. If $Q_k \notin H_0$ but there is a tuple $(T_1, T_2, T_3, T_4) \in \mathsf{DE}$-table such that $T_1 = y, T_2 = m, T_3 = \mathsf{str}||0^{n_3}$ and $T_4 = \mathsf{PE}(y, m, r)$, then the system responds with str, else checks the following cases;
- Case 3. If $Q_k \notin H_0$ but there is a tuple $(T_1, T_2, T_3, T_4) \in \mathsf{DD}$-table such that $T_3 = m, \mathsf{PKG}(T_1) = y, \mathsf{PE}(y, m, r) = T_2$ and $T_4 = \mathsf{str}||0^{n_3}$, then the system reponds with str, else checks the following cases;
- Case 4. Otherwise, the system randomly samples $\mathsf{str} \leftarrow \{0,1\}^{n_3}$, responds with str and inserts $(y, m, \mathsf{str}||0^{n_3}, \mathsf{PE}(y, m, r))$ into DE-table.

Game 3. This game is identical to Game 2, except that to answer DE queries, where the system only uses the tables and honest interfaces. Assuming $(y, m, r_1||r_2, \mathsf{DE})$ is the k-th query, then

- Case 1. If $Q_k \in \mathsf{DE}$, then it responds with the corresponding output string;
- Case 2. If $Q_k \notin \mathsf{DE}$ and $r_2 \neq 0^{n_3}$, then
  1. If there is a tuple $T = (T_1, T_2, T_3, T_4)$ in DD-table such that $y = \mathsf{PKG}(T_1)$, $T_3 = m$ and $T_4 = r$, then the system responds with T_2;
  2. Otherwise the system responds with $c \leftarrow \mathcal{C}$.
- Case 3. If $Q_k \notin \mathsf{DE}$ and $r_2 = 0^{n_3}$, then
  1. If there is a tuple $T = (T_1, T_2, T_3, T_4)$ in DD-table such that $y = \mathsf{PKG}(T_1)$, $T_3 = m$ and $T_4 = r$, then the system responds with T_2;
  2. If there is $(T_1, T_2, T_3, T_4) \in H_0$ such that $T_1 = y, T_2 = m, T_4 = r||0^{n_3}$, then the system responds with $\mathsf{PE}(y, m, T_3)$;
  3. Otherwise the system samples $r^* \leftarrow \{0,1\}^{2n_3}$ and responds with $\mathsf{PE}(y, m, r^*)$.

Game 4. This game is identical to Game 1, except that to answer DD queries, where the system only uses the tables and honest interfaces. Assuming (x, c, DD) is the k-th query, then

- Case 1. If $Q_k \in \mathsf{DD}$, then it responds with the corresponding output string;

- Case 2. If $Q_k \notin \mathsf{DD}$ but there is a tuple $(T_1, T_2, T_3, T_4) \in \mathsf{DE}$ such that $T_1 = \mathsf{PKG}(x), T_4 = c$, then the system responds with (T_2, T_3);
- Case 3. Otherwise, the system calls $m^* = (x, c, \mathsf{PD})$, and responds as follows:
  1. If $m^* = \perp$, then the system aborts.
  2. Else if there is a tuple $(T_1, T_2, T_3, T_4) \in H_0$ such that $T_1 = \mathsf{PKG}(x) = T_1, T_2 = m^*$ and $\mathsf{PE}(T_1, T_2, T_3) = c$, then the system responds with $(\mathsf{str}_1, \mathsf{str}_2) = (T_2, T_4||0^{n_3})$;
  3. Otherwise, the system samples $\mathsf{str} \leftarrow \{0,1\}^{n_3}$ and responds with $(m^*, \mathsf{str}||0^{n_3})$.

Game 5. In Game 4, the queries to $H_0, \mathsf{DKG}, \mathsf{DE}, \mathsf{DD}$ are answered by the tables which're maintained by the system and by making queries to $\mathsf{PKG}, \mathsf{PE}, \mathsf{PD}$. The system never makes queries directly to $\tilde{H}_0, \tilde{\mathsf{DKG}}, \tilde{\mathsf{DE}}$ and $\tilde{\mathsf{DD}}$; these oracles are *only* used to answer the $\mathsf{PKG}, \mathsf{PE}, \mathsf{PD}$ queries (either generated by the adversary or by the system's response to $H_0, \mathsf{DKG}, \mathsf{DE}, \mathsf{DD}$ queries). At this point, it is straightforward to show that we can replace $\mathsf{PKG}, \mathsf{PE}$ and PD with the ideal versions from Definition 6, resulting in Game 5.

We note that in Game 5, the system is efficient, and it responds to the adversarial interfaces just by keeping several tables and accessing the honest interfaces. Thus, we can build a simulator that responds to $H_0, \mathsf{DKG}, \mathsf{DE}, \mathsf{DD}$ queries exactly as the system does in Game 5. The result is that Game 5 corresponds to the ideal world. Since Game 0 is the real world, it suffices to prove that any adjacent games are indistinguishable. Next we will give the description of our simulator and prove the indistinguishability between each adjacent games.

Simulator. Let $\Pi_{\mathsf{PKE}} = (\mathsf{PKG}, \mathsf{PE}, \mathsf{PD})$ be an ideal PKE, associated with secret key space $\mathcal{X}$, public key space $\mathcal{Y} = \{0,1\}^{n_1}$, message space $\mathcal{M} = \{0,1\}^{n_2}$, nonce space $\mathcal{R} = \{0,1\}^{2n_3}$ and ciphertext space $\mathcal{C}$, where 1)$|\mathcal{X}| \leq |\mathcal{Y}|$; 2) $|\mathcal{Y} \times \mathcal{M} \times \mathcal{R}| \leq |\mathcal{C}|$. The simulator $\mathcal{S}$ responds the queries as follows:

H_0-QUERY. Suppose (y, m, r, H_0) is the k-th query, the system responds as follows:

- Case 1. If $Q_k \in H_0$, then it responds with the corresponding output string;
- Case 2. If $Q_k \notin H_0$ but there is a tuple $(T_1, T_2, T_3, T_4) \in \mathsf{DE}$-table such that $T_1 = y, T_2 = m, T_3 = \mathsf{str}||0^{n_3}$ and $T_4 = \mathsf{PE}(y, m, r)$, then the system responds with str, else checks the following cases;
- Case 3. If $Q_k \notin H_0$ but there is a tuple $(T_1, T_2, T_3, T_4) \in \mathsf{DD}$-table such that $T_3 = m, \mathsf{PKG}(T_1) = y, \mathsf{PE}(y, m, r) = T_2$ and $T_4 = \mathsf{str}||0^{n_3}$, then the system reponds with str, else checks the following cases;
- Case 4. Otherwise, the system randomly samples $\mathsf{str} \leftarrow \{0,1\}^{n_3}$, responds with str and inserts $(y, m, \mathsf{str}||0^{n_3}, \mathsf{PE}(y, m, r))$ into DE-table.

DKG-QUERY. Suppose (x, DKG) is the k-th query, the system responds with $\mathsf{PKG}(x)$.

DE-QUERY. Suppose (y, m, r, DE) is the k-th query(for ease, we split $r = r_1||r_2$), the system responds as follows:

- Case 1. If $Q_k \in \mathsf{DE}$, then it responds with the corresponding output string;
- Case 2. If $Q_k \notin \mathsf{DE}$ and $r_2 \neq 0^{n_3}$, then
  1. If there is a tuple $T = (T_1, T_2, T_3, T_4)$ in DD-table such that $y = \mathsf{PKG}(T_1)$, $T_3 = m$ and $T_4 = r$, then the system responds with T_2;
  2. Otherwise the system responds with $c \leftarrow \mathcal{C}$.
- Case 3. If $Q_k \notin \mathsf{DE}$ and $r_2 = 0^{n_3}$, then

1. If there is a tuple $T = (T_1, T_2, T_3, T_4)$ in DD-table such that $y = \mathsf{PKG}(T_1)$, $T_3 = m$ and $T_4 = r$, then the system responds with T_2;
2. If there is $(T_1, T_2, T_3, T_4) \in H_0$ such that $T_1 = y, T_2 = m, T_4 = r||0^{n_3}$, then the system responds with $\mathsf{PE}(y, m, T_3)$;
3. Otherwise the system samples $r^* \leftarrow \{0,1\}^{2n_3}$ and responds with $\mathsf{PE}(y, m, r^*)$.

DD-QUERY. Suppose (x, c, DD) is the k-th query, the system responds as follows:

– Case 1. If $Q_k \in \mathsf{DD}$, then it responds with the corresponding output string;
– Case 2. If $Q_k \notin \mathsf{DD}$ but there is a tuple $(T_1, T_2, T_3, T_4) \in \mathsf{DE}$ such that $T_1 = \mathsf{PKG}(x), T_4 = c$, then the system responds with (T_2, T_3);
– Case 3. Otherwise, the system calls $m^* = (x, c, \mathsf{PD})$, and responds as follows:
  1. If $m^* = \perp$, then the system aborts.
  2. Else if there is a tuple $(T_1, T_2, T_3, T_4) \in H_0$ such that $T_1 = \mathsf{PKG}(x) = T_1, T_2 = m^*$ and $\mathsf{PE}(T_1, T_2, T_3) = c$, then the system responds with $(\mathsf{str}_1, \mathsf{str}_2) = (T_2, T_4||0^{n_3})$;
  3. Otherwise, the system samples $\mathsf{str} \leftarrow \{0,1\}^{n_3}$ and responds with $(m^*, \mathsf{str}||0^{n_3})$.

Note that $\mathcal{S}$ works exactly the same as the system in Game 5. Moreover, the distribution of the honest interfaces in Game 8 is also identical to the ones in the ideal game, which means, for any adversary, the advantage in both Game 5 and ideal game is identical. Therefore, it suffices to prove that any adjacent games are indistinguishable.

Claim. Game 0 $\approx$ Game 1.

Proof. It's trivial to note that Pr[Game 0] = Pr[Game 1], as in Game 1, the system only changes the way of maintaining the tables and the terms stored in each tuple is identical to the responses of real oracles.

Claim. Game 1 $\approx$ Game 2.

Proof. Firstly, we note that, in either Game 1 or Game 2, the adversary's view on the honest interfaces is consistent, hence it suffices to prove that, with high probability, the view on the adversarial interfaces also does not change. According to the definition, the only difference between Game 1 and Game 2 is when Case 2.3 occurs. Suppose $(\mathsf{PK}, \mathsf{M}, \mathsf{R}, H_0)$ is the k-th query, the adversary's view on the adversarial interfaces are as follows:

View on H_0. By definition, $\tilde{H}_0$ is a random oracle such that 1) the response is consistent; 2) the response to a fresh input is uniformly random.

Thus, the adversary's view on H_0 is consistent in both games, if the responses of H_0 queries satisfy the two properties above. When case 2.3 occurs, we note that, after Q_k in Game 2, we implicitly set

$$H_0(\mathsf{PK}, \mathsf{M}, \mathsf{R}) = \mathsf{r} \text{ where } r \xleftarrow{\$} \{0,1\}^{n_3}$$

as $(\mathsf{PK}, \mathsf{M}, \mathsf{R}, H_0)$ never appears in previous $k-1$ queries, property 1 and 2 preserves.

View on DKG. Recalling that $(\mathsf{SK}, \mathsf{DKG}) = \mathsf{PKG}(\mathsf{SK})$, the view on DKG would be always consistent.

View on DE. Due to definition, $\tilde{\mathsf{DE}}$ is an encryption algorithm such that 1) the response is consistent; 2) the response to a fresh input is uniformly random; 3) the encryption is an injection; 4) $\mathsf{DE}(\mathsf{PK}, \mathsf{M}, (\mathsf{PK}, \mathsf{M}, \mathsf{R}, H_0)||0^{n_3}) = \mathsf{PE}(\mathsf{PK}, \mathsf{M}, \mathsf{R})$.

Thus, the adversary'view on DE is consistent in both games, if the responses of the DE-queries satisfy the four properties above. When case 2.3 occurs, we note that, after Q_k in Game 2, we implicitly set

$$\mathsf{DE}(\mathsf{PK},\mathsf{M},r||0^{n_3}) = \mathsf{PE}(\mathsf{PK},\mathsf{M},\mathsf{R}) = \tilde{\mathsf{DE}}(\mathsf{PK},\mathsf{M},\tilde{H}_0(\mathsf{PK},\mathsf{M},\mathsf{R})||0^{n_3}))$$

hence, as long no query $(\mathsf{PK},\mathsf{M},r||0^{n_3},\mathsf{DE})$ appears in the previous $k-1$ queries, property 1 and 4 preserves. Moreover, as $\tilde{H}_0$ is a random oracle, and if $\tilde{H}_0(\mathsf{PK},\mathsf{M},\mathsf{R})$ never appears in the previous $k-1$ queries, then $\tilde{\mathsf{DE}}(\mathsf{PK},\mathsf{M},\tilde{H}_0(\mathsf{PK},\mathsf{M},\mathsf{R})||0^{n_3}))$ is uniformly random. For property 3, we note that, in Game 2,

$$\mathsf{DE}(\mathsf{PK},\mathsf{M},r||0^{n_3}) = \mathsf{DE}(\mathsf{PK},\mathsf{M},\tilde{H}_0(\mathsf{PK},\mathsf{M},\mathsf{R})||0^{n_3})$$

hence if if $\tilde{H}_0(\mathsf{PK},\mathsf{M},\mathsf{R})$ never appears after Q_k, the responses of DE-query are injective.

View on DD. According to the definition, $\tilde{\mathsf{DD}}$ is a decryption algorithm such that 1) the response is consistent; 2) $\mathsf{DD}(\mathsf{SK},\mathsf{DE}(\mathsf{PK},\mathsf{M},\mathsf{r})) = (\mathsf{M},\mathsf{r})$; 3) if non-abort, then $\mathsf{DD}(\mathsf{SK},\mathsf{C}) = \mathsf{DD}(\mathsf{SK},\mathsf{C}') \Rightarrow \mathsf{C} = \mathsf{C}'$; 4) if $\mathsf{DD}(\mathsf{SK},\mathsf{C}) = (\mathsf{M},\mathsf{r}_1||0^{\mathsf{n}_3})$ then $\mathsf{PD}(\mathsf{SK},\mathsf{C}) = \mathsf{M}$; 5) if $\mathsf{DD}(\mathsf{SK},\mathsf{C}) = (\mathsf{M},\mathsf{r}_1||\mathsf{r}_2)$ where $\mathsf{r}_2 \neq 0^{\mathsf{n}_3}$, then $\mathsf{PD}(\mathsf{SK},\mathsf{C}) = \perp$.

Hence, the adversary's view on DD is consistent in both games, if the responses of the DD-queries satisfy the five properties above. When case 2.3 occurs, we note that, after Q_k in Game 2, we implicitly set

$$\mathsf{DD}(\mathsf{SK},\mathsf{DE}(\mathsf{PK},\mathsf{M},\tilde{H}_0(\mathsf{PK},\mathsf{M},\mathsf{R}))) = \mathsf{DD}(\mathsf{SK},\mathsf{DE}(\mathsf{PK},\mathsf{M},\mathsf{r})) = (\mathsf{M},\mathsf{r}||0^{\mathsf{n}_3})$$

as $(\mathsf{SK},\mathsf{DE}(\mathsf{PK},\mathsf{M},\tilde{H}_0(\mathsf{PK},\mathsf{M},\mathsf{R})),\mathsf{DD})$ never appears, the response of DD satisfies property 1, 2, 4, 5. For property 3, we have that, if $\tilde{H}_0(\mathsf{PK},\mathsf{M},\mathsf{R}') = r$

$$\mathsf{DD}(\mathsf{SK},\mathsf{PE}(\mathsf{PK},\mathsf{M},\tilde{H}_0(\mathsf{PK},\mathsf{M},\mathsf{R}'))) = (\mathsf{M},\mathsf{r})$$

hence if no R' is queried such that $\tilde{H}_0(\mathsf{PK},\mathsf{M},\mathsf{R}') = \mathsf{r}$, then property 3 holds.

Now, we can easily bound the union of these bad events as:

$$\Pr[\mathsf{Bad}] \leq \frac{k-1}{2^{n_3}} + \frac{k-1}{2^{n_3}} + \frac{q-k}{2^{n_3}} + \frac{q}{2^{n_3}} \leq \frac{3q}{2^{n_3}}.$$

which referring to

$$|\Pr[\text{Game 1}] - \Pr[\text{Game 2}]| \leq q \cdot \Pr[\mathsf{Bad}] \leq \frac{3q^2}{2^{n_3}} \leq \mathsf{negl}(\lambda).$$

Claim. Game 2 $\approx$ Game 3.

Proof. Firstly, it's trivial that, in either Game 2 or Game 3, the adversary's view on the honest interfaces is consistent, hence it suffices to prove that, with high probability, the view on the adversarial interfaces also does not change. According to the definition, there are two differences between Game 2 and Game 3, case 2.2 and case 3.3. In the following, we analyze the view on adversarial interfaces by case.

Case 2.2. Suppose $(\mathsf{PK},\mathsf{M},\mathsf{r}_1||\mathsf{r}_2,\mathsf{DE})$ is the k-th query, where $r_2 \neq 0^{n_3}$.

View on H_0. Under case 2.2, the responses of H_0 queries do not change, thus the view preserves.

View on DKG. Same as above.

View on DE. Due to definition, $\tilde{\mathsf{DE}}$ is an encryption algorithm such that 1) the response is consistent; 2) the response to a fresh input is uniformly random; 3) the encryption is an injection; 4) $\mathsf{DE}(\mathsf{PK}, \mathsf{M}, (\mathsf{PK}, \mathsf{M}, \mathsf{R}, H_0)||0^{n_3}) = \mathsf{PE}(\mathsf{PK}, \mathsf{M}, \mathsf{R})$.

Thus, the adversary'view on DE is consistent in both games, if the responses of the DE-queries satisfy the four properties above. When case 2.2 occurs, we implicitly set

$$\mathsf{DE}(\mathsf{PK}, \mathsf{M}, \mathsf{r}_1||\mathsf{r}_2) = \mathsf{C}, \text{ where } \mathsf{C} \leftarrow \mathcal{C}.$$

Property 1, 2 and 4 easily follows, as long as no $(\mathsf{PK}', \mathsf{M}', \mathsf{r}_1'||\mathsf{r}_2')$ such that $\mathsf{DE}(\mathsf{PK}', \mathsf{M}', \mathsf{r}_1'||\mathsf{r}_2') = \mathsf{C}$, property 3 holds.

View on DD. According to the definition, $\tilde{\mathsf{DD}}$ is a decryption algorithm such that 1) the response is consistent; 2) $\mathsf{DD}(\mathsf{SK}, \mathsf{DE}(\mathsf{PK}, \mathsf{M}, \mathsf{r})) = (\mathsf{M}, \mathsf{r})$; 3) if non-abort, then $\mathsf{DD}(\mathsf{SK}, \mathsf{C}) = \mathsf{DD}(\mathsf{SK}, \mathsf{C}') \Rightarrow \mathsf{C} = \mathsf{C}'$; 4) if $\mathsf{DD}(\mathsf{SK}, \mathsf{C}) = (\mathsf{M}, \mathsf{r}_1||0^{n_3})$ then $\mathsf{PD}(\mathsf{SK}, \mathsf{C}) = \mathsf{M}$; 5) if $\mathsf{DD}(\mathsf{SK}, \mathsf{C}) = (\mathsf{M}, \mathsf{r}_1||\mathsf{r}_2)$ where $\mathsf{r}_2 \neq 0^{n_3}$, then $\mathsf{PD}(\mathsf{SK}, \mathsf{C}) = \perp$.

Hence, the adversary's view on DD is consistent in both games, if the responses of the DD-queries satisfy the five properties above. When case 2.2 occurs, we note that, after Q_k in Game 2, we implicitly set

$$\mathsf{DD}(\mathsf{SK}, \mathsf{C}) = \mathsf{DD}(\mathsf{DE}(\mathsf{PK}, \mathsf{M}, \mathsf{r}_1||\mathsf{r}_2)) = (\mathsf{M}, \mathsf{r}_1||\mathsf{r}_2).$$

as $(\mathsf{SK}, \mathsf{C}, \mathsf{DD})$ never appears, the responses of DD satisfy property 1, 2, 4. For property 3, we have:

$$\mathsf{DD}(\mathsf{SK}, \tilde{\mathsf{DE}}(\mathsf{PK}, \mathsf{M}, \mathsf{r}_1||\mathsf{r}_2) = (\mathsf{M}, \mathsf{r}_1||\mathsf{r}_2)),$$

if $\tilde{\mathsf{DE}}(\mathsf{PK}, \mathsf{M}, \mathsf{r}_1||\mathsf{r}_2)$ never appears, property 3 holds. Moreover, if and only if there exist $R \in \{0,1\}^{n_3}$ such that $\mathsf{C} = \mathsf{PE}(\mathsf{PK}, \mathsf{M}, \mathsf{R})$, PD would not abort.

Case 3.3. Suppose $(\mathsf{PK}, \mathsf{M}, \mathsf{r}||0^{n_3})$ is the k-th query.

View on H_0. By definition, $\tilde{H_0}$ is a random oracle such that 1) the response is consistent; 2) the response to a fresh input is uniformly random.

Thus, the adversary's view on H_0 is consistent in both games, if the responses of H_0 queries satisfy the two properties above. When case 3.3 occurs, let $\mathsf{R}^* \xleftarrow{\$} \{0,1\}^{n_3}$, the system sets

$$\mathsf{DE}(\mathsf{PK}, \mathsf{M}, \mathsf{r}||0^{n_3}) = \mathsf{PE}(\mathsf{PK}, \mathsf{M}, \mathsf{R}^*)$$

hence after Q_k, it implicitly sets

$$H_0(\mathsf{PK}, \mathsf{M}, \mathsf{R}) = \mathsf{r}, \text{ if } \tilde{H_0}(\mathsf{PK}, \mathsf{M}, \mathsf{R}) = \tilde{H_0}(\mathsf{PK}, \mathsf{M}, \mathsf{R}^*)$$

We observe that r is chosen by the adversary, so r is not uniformly distributed. However, if during the queries, no such R appears, the view on H_0 would not change. In fact, in either Game 2 or Game 3, $\tilde{H_0}$ is never used, which means, except with random guessing, such a R never appears.

View on DKG. Same as above.

View on DE. Due to definition, $\tilde{\mathsf{DE}}$ is an encryption algorithm such that 1) the response is consistent; 2) the response to a fresh input is uniformly random; 3) the encryption is an injection; 4) $\mathsf{DE}(\mathsf{PK}, \mathsf{M}, (\mathsf{PK}, \mathsf{M}, \mathsf{R}, H_0)||0^{n_3}) = \mathsf{PE}(\mathsf{PK}, \mathsf{M}, \mathsf{R})$.

Thus, the adversary'view on DE is consistent in both games, if the responses of the DE-queries satisfy the four properties above. When case 2.2 occurs, we implicitly set

$$\mathsf{DE}(\mathsf{PK}, \mathsf{M}, \mathsf{r}||0^{n_3}) = \mathsf{PE}(\mathsf{PK}, \mathsf{M}, \mathsf{R}^*) = \tilde{\mathsf{DE}}(\mathsf{PK}, \mathsf{M}, \tilde{H_0}(\mathsf{PK}, \mathsf{M}, \mathsf{R}^*)||0^{n_3}),$$

if $\tilde{H}_0(\mathsf{PK}, \mathsf{M}, \mathsf{R}^*)$ never appears in the previous query, then property 1, 2, 4 hold. For property 3, if $\tilde{H}_0(\mathsf{PK}, \mathsf{M}, \mathsf{R}^*)$ never appears after Q_k and for query $\mathsf{DE}(\mathsf{PK}, \mathsf{M}, \mathsf{r}'||0^{\mathsf{n}_3})$, the system does not set it to be $\mathsf{PE}(\mathsf{PK}, \mathsf{M}, \mathsf{R}^*)$, then the responses are injective.

View on DD. According to the definition, $\tilde{\mathsf{DD}}$ is a decryption algorithm such that 1) the response is consistent; 2) $\mathsf{DD}(\mathsf{SK}, \mathsf{DE}(\mathsf{PK}, \mathsf{M}, \mathsf{r})) = (\mathsf{M}, \mathsf{r})$; 3) if non-abort, then $\mathsf{DD}(\mathsf{SK}, \mathsf{C}) = \mathsf{DD}(\mathsf{SK}, \mathsf{C}') \Rightarrow \mathsf{C} = \mathsf{C}'$; 4) if $\mathsf{DD}(\mathsf{SK}, \mathsf{C}) = (\mathsf{M}, \mathsf{r}_1||0^{\mathsf{n}_3})$ then $\mathsf{PD}(\mathsf{SK}, \mathsf{C}) = \mathsf{M}$; 5) if $\mathsf{DD}(\mathsf{SK}, \mathsf{C}) = (\mathsf{M}, \mathsf{r}_1||\mathsf{r}_2)$ where $\mathsf{r}_2 \neq 0^{\mathsf{n}_3}$, then $\mathsf{PD}(\mathsf{SK}, \mathsf{C}) = \perp$.

Hence, the adversary's view on DD is consistent in both games, if the responses of the DD-queries satisfy the five properties above. When case 3.3 occurs, we note that, after Q_k in Game 2, we implicitly set

$$\mathsf{DD}(\mathsf{SK}, \mathsf{C}) = \mathsf{DD}(\mathsf{DE}(\mathsf{PK}, \mathsf{M}, \tilde{H}_0(\mathsf{PK}, \mathsf{M}, \mathsf{R}^*)||0^{n_3})) = (\mathsf{M}, \mathsf{r}||0).$$

if C does not appears in the previous $k-1$ queries, then the responses satisfy property 1, 2, 4, 5. For property 3, we have that

$$\mathsf{DD}(\mathsf{SK}, \mathsf{PE}(\mathsf{PK}, \mathsf{M}, \mathsf{R}_\mathsf{r})) = (\mathsf{M}, \mathsf{r}||0^{\mathsf{n}_3}), \text{ where } \tilde{H}_0(\mathsf{PK}, \mathsf{M}, \mathsf{R}_\mathsf{r}) = r.$$

Note that in either Game 2 or Game 3, $\tilde{H}_0$ is hidden, which means, expect with random guessing, $\mathsf{PE}(\mathsf{PK}, \mathsf{M}, \mathsf{R}_\mathsf{r})) = (\mathsf{M}, \mathsf{r}||0^{\mathsf{n}_3})$ never appears.

Now we can easily bound the union of these bad events as:

$$\Pr[\mathsf{Bad}] \leq \frac{q}{|\mathcal{C}|} + \frac{q \cdot 2^{2n_3}}{|\mathcal{C}|} + \frac{q2^{n_3}}{2^{2n_3}} + \frac{q}{2^{n_3}} + \frac{k-1}{2^{n_3}} + \frac{q}{2^{n_3}} \leq \frac{2q}{2^{n_1+n_2}} + \frac{4q}{2^{n_3}}$$

which referring to

$$|\Pr[\text{Game } 2] - \Pr[\text{Game } 3]| \leq q \cdot \Pr[\mathsf{Bad}] \leq \mathsf{negl}(\lambda).$$

Claim. Game 3 $\approx$ Game 4.

Proof. Firstly, we note that, in either Game 3 or Game 4, the adversary's view on the honest interfaces is consistent, hence it suffices to prove that, with high probability, the view on the adversarial interfaces also does not change. According to the definition, the only difference between Game 1 and Game 2 is when Case 3.3 occurs. Suppose $(\mathsf{SK}, \mathsf{C}, H_0)$ is the k-th query, the adversary's view on the adversarial interfaces are as follows:

View on H_0. By definition, $\tilde{H}_0$ is a random oracle such that 1) the response is consistent; 2) the response to a fresh input is uniformly random.

Thus, the adversary's view on H_0 is consistent in both games, if the responses of H_0 queries satisfy the two properties above. When case 3.3 occurs, let $\mathsf{r} \xleftarrow{\$} \{0,1\}^{n_3}$, the system sets:

$$\mathsf{DD}(\mathsf{SK}, \mathsf{C}) = (\mathsf{M}, \mathsf{r}||0^{\mathsf{n}_3})$$

where $\mathsf{C} = \mathsf{PE}(\mathsf{PK}, \mathsf{M}, \mathsf{R})$ and R might be known by the adversary. Then after Q_k, we implicitly set $\tilde{H}_0(\mathsf{PK}, \mathsf{M}, \mathsf{R}) = r$. As r is randomly chosen and $(\mathsf{PK}, \mathsf{M}, \mathsf{R}, H_0)$ never appears in previous queries, property 1 and 2 preserve.

View on DKG. Same as above.

View on DE. Due to definition, $\tilde{\mathsf{DE}}$ is an encryption algorithm such that 1) the response is consistent; 2) the response to a fresh input is uniformly random; 3) the encryption is an injection; 4) $\mathsf{DE}(\mathsf{PK}, \mathsf{M}, (\mathsf{PK}, \mathsf{M}, \mathsf{R}, H_0)||0^{n_3}) = \mathsf{PE}(\mathsf{PK}, \mathsf{M}, \mathsf{R})$.

Thus, the adversary'view on DE is consistent in both games, if the responses of the DE-queries satisfy the four properties above. When case 3.3 occurs, we note that, after Q_k in Game 4, we implicitly set

$$\mathsf{DE}(\mathsf{PK},\mathsf{M},\mathsf{r}||0^{\mathsf{n}_3}) = C = \tilde{\mathsf{DE}}(\mathsf{PK},\mathsf{M},\tilde{H}_0(\mathsf{PK},\mathsf{M},\mathsf{R})||0^{n_3}).$$

If r never appears in previous queries, then property 1 and 4 hold. Moreover, although R is chosen by adversary, $\tilde{H}_0(\mathsf{PK},\mathsf{M},\mathsf{R})$ is hidden and if $\tilde{H}_0(\mathsf{PK},\mathsf{M},\mathsf{R})$ never appears in the previous queries, $\mathcal{C}$ is a fresh value, which means property 2 holds.

Besides, we note that in either Game 3 or Game 4, DE is responded by the tables or randomly, hence, except with negligible probability, the responses are injective.

View on DD. According to the definition, $\tilde{\mathsf{DD}}$ is a decryption algorithm such that 1) the response is consistent; 2) $\mathsf{DD}(\mathsf{SK},\mathsf{DE}(\mathsf{PK},\mathsf{M},\mathsf{r})) = (\mathsf{M},\mathsf{r})$; 3) if non-abort, then $\mathsf{DD}(\mathsf{SK},\mathsf{C}) = \mathsf{DD}(\mathsf{SK},\mathsf{C}') \Rightarrow \mathsf{C} = \mathsf{C}'$; 4) if $\mathsf{DD}(\mathsf{SK},\mathsf{C}) = (\mathsf{M},\mathsf{r}_1||0^{\mathsf{n}_3})$ then $\mathsf{PD}(\mathsf{SK},\mathsf{C}) = \mathsf{M}$; 5) if $\mathsf{DD}(\mathsf{SK},\mathsf{C}) = (\mathsf{M},\mathsf{r}_1||\mathsf{r}_2)$ where $\mathsf{r}_2 \neq 0^{\mathsf{n}_3}$, then $\mathsf{PD}(\mathsf{SK},\mathsf{C}) = \perp$.

Hence, the adversary's view on DD is consistent in both games, if the responses of the DD-queries satisfy the five properties above. When case 3.3 occurs, we note that, after Q_k in Game 4,$\mathsf{DD}(\mathsf{SK},\mathsf{C}) = (\mathsf{M},\mathsf{r}||0^{\mathsf{n}_3})$. It's trivial to note that property 1, 2, 4 and 5 hold. Moreover, unless we randomly sample r twice for different DD queries, property 3 holds.

Now we can easily bound the union of these bad events as:

$$\Pr[\mathsf{Bad}] \leq \frac{k-1}{2^{n_3}} + \frac{q}{2^{n_3}} + \frac{q}{2^{n_3}} \leq \frac{4q}{2^{n_3}}$$

which referring to

$$|\Pr[\text{Game 3}] - \Pr[\text{Game 4}]| \leq q \cdot \Pr[\mathsf{Bad}] \leq \mathsf{negl}(\lambda).$$

Claim. Game 4 $\approx$ Game 5.

Proof. Due to the definition, we note that in Game 4, the system responds all of the adversarial interfaces just using tables and honest interfaces, it never directly calls the real oracles. Besides, $(\tilde{\mathsf{DKG}},\tilde{\mathsf{DE}},\tilde{\mathsf{DD}})$ is an ideal DPKE scheme and $\tilde{H}_0$ is a random oracle, the only chance that adversary can differ Game 4 and Game 5 is one of the following events occurs:

- Case 1. There are queries $(\mathsf{PK},\mathsf{M},\mathsf{R}_1,\mathsf{PE})$ and $((\mathsf{PK},\mathsf{M},\mathsf{R}_1,\mathsf{PE}))$ such that $\tilde{H}_0(\mathsf{PK},\mathsf{M},\mathsf{R}_1) = \tilde{H}_0(\mathsf{PK},\mathsf{M},\mathsf{R}_2)$;
- Case 2. There is a $\mathsf{r} \in \{0,1\}^{n_3}$ and (PK,M) such that r has no pre-image, which means $\forall \mathsf{R} \in \{0,1\}^{2n_3}, \tilde{H}_0(\mathsf{PK},\mathsf{M},\mathsf{R}) \neq \mathsf{r}$.

And it's trivial to note that if none of the cases occurs then we can replace the honest interfaces $(\mathsf{PKG},\mathsf{PE},\mathsf{PD})$ with an ideal PKE. Moreover we can bound the probability of the bad cases as:

$$\Pr[\text{Case 1}] \leq \frac{q^2}{2^{n_3}}, \Pr[\text{Case 2}] \leq (1 - \frac{1}{2^{n_3}})^{2^{2n_3}} \leq (\frac{1}{e})^{2^{n_3}}.$$

which refers to

$$|\Pr[\text{Game 4}] - \Pr[\text{Game 5}]| \leq \mathsf{negl}(\lambda).$$

Combining together, we complete the whole proof. □

10 Proof of Theorem 10

In this section, we give the full proof of Theorem 10, that we show Π_{Sig} is indifferentiable from an ideal signature scheme.

Proof. According to the definition of indifferentiability, we immediately observe that the adversary has three honest interfaces ($\mathsf{SKG}, \mathsf{SS}, \mathsf{SV}$) and six adversarial interfaces (H_0, H_1, P,P^{-1}, E, E^{-1}). Therefore, we need to build an efficient simulator $\mathcal{S}$ that can simulate the six adversarial interfaces H_0, H_1, P, P^{-1}, E and E^{-1} properly, which means, for any PPT differentiator $\mathcal{D}$, the view of $\mathcal{D}$ in the real game is computationally close to the view in the ideal game. To do so, we will go through with a sequence of hybrid games. Before the description of the games, we first specify some parameters and events:

- there are nine types of query: $(x, H_0), (m, H_1), (y, P), (y, P^{-1}), (y, m, z, E), (y, m, z, E^{-1})$, $(x, \mathsf{SKG}), (x, m, \mathsf{SS}), (y, m, \Sigma, \mathsf{SV}$ where $x \leftarrow \mathcal{X}, y \leftarrow \mathcal{Y}, m \leftarrow \mathcal{M}, z \leftarrow \mathcal{Z}$;
- adversary makes at most q queries to the system, where $q = \mathsf{poly}(\lambda)$;
- the real oracles used in the construction are $\tilde{H}_0, \tilde{H}_1, \tilde{P}, \tilde{P}^{-1}, \tilde{E}, \tilde{E^{-1}}$;
- the advantage of random-message attack is bounded by ϵ_1;
- the advantage of pseudorandom public key is bounded by ϵ_2.

Note that in the normal lazy sampling of random oracles or permutations, each output will be chosen essentially at random. However, a simulator for indifferentiability will need to occasionally sample in such a way as to be consistent with the ideal signature. Next we define two events, named "Strong Consistency Check"(SCC) and "Weak Consistency Check"(WCC). These checks look for the cases where the adversary may already have the necessary information to predict the query response without making the query. If the checks pass, it means the adversary is unable to make such a prediction, and the simulator is essentially free to answer randomly. On the other hand, if the checks fail, the simulator must answer carefully to be consistent with the adversary's prediction.

Strong Consistency Check. Let $Q_1, \ldots, Q_q$ be the sequence of the queries, we say event SCC occurs for the k-th query ($\mathsf{SCC}_k = 1$), if any one of the following cases is satisfied:

- Case 1. The k-th query is P query, say (y, P) and in the previous $k-1$ queries, there is no query with form of $(x, H_0), (x, \mathsf{SKG})$ or (x, m, SS) such that $\mathsf{SKG}(x) = y$.
  Note that if there had been a previous query on such an x, then it would be possible for the adversary to predict $P(y) = P(\mathsf{SKG}(x))$ without making a P query at all by just evaluating $\mathsf{keygen}(H_0(x))$.
- Case 2. The k-th query is P^{-1} query, say (y, P^{-1}) and in the previous $k-1$ queries, there exists no query with form of $(x, H_0), (x, \mathsf{SKG})$ or (x, m, SS) such that $\mathsf{keygen}(\tilde{H}_0(x)) = y$. Note that if there had been a previous query on such an x, the adversary can predict $P^{-1}(y) = P^{-1}(\mathsf{keygen}(\tilde{H}_0(x)))$ by querying $\mathsf{SKG}(x)$.

We note in the ideal world, the simulator is unable to tell if SCC happens, since doing so requires knowing the adversary's queries to $\mathsf{SKG}, \mathsf{SS}$ and SV. Instead, the simulator will be able to carry out a weak consistency check, WCC:

Weak Consistency Check. Let $Q_1, \ldots, Q_q$ be the sequence of the queries, we say event WCC occurs for the k-th query ($\mathsf{WCC}_k = 1$), if one of the following case satisfies:

- Case 1. The k-th query is P query, say (y, P) and in the previous $k-1$ queries, there is no query with form of (x, H_0) such that $\mathsf{SKG}(x) = y$.

- Case 2. The k-th query is P^{-1} query, say (y, P^{-1}) and in the previous $k-1$ queries, there is no query with form of (x, H_0) such that $\mathsf{keygen}(\tilde{H}_0(x)) = y$.
- Case 3. The k-th query is E query, say (y, m, z, E) and in the previous $k-1$ queries, there is no (m, H_1) query or no query with form of (x, H_0) such that $\mathsf{SKG}(x) = y$.
- Case 4. The k-th query is E^{-1} query, say (y, m, z, E^{-1}) and in the previous $k-1$ queries, there is no (m, H_1) query or no query with form of (x, H_0) such that $\mathsf{SKG}(x) = y$.

This weak consistency check will not catch all the bad cases, but we will demonstrate that the adversary is unable to generate such bad cases, except with negligible probability.

Now we are ready the define the games. After each game, we also give the intuition for why that game is indistinguishable from the previous game.

Game 0. This game is identical to the real game, where every query is answered by applying real oracles. For instance, the response of (x, H_0) is $\tilde{H}_0(x)$, the response of (x, SKG) is $\tilde{P}^{-1}(\mathsf{keygen}(\tilde{H}_0(x)))$ and so forth. Moreover, during the queries, it also maintains six tables, referring to H_0-table, H_1-table, P-table, P^{-1}-table, E-table and E^{-1}-table. Concretely,

- H_0-table: Initially empty, consists of tuples with form of (x, r, y, y'). Once the adversary makes a (x, H_0) query which does not exist in H_0-table (no tuple that the first element of it is x), it inserts $(x, \tilde{H}_0(x), \mathsf{keygen}(\tilde{H}_0(x)),$
$\mathsf{SKG}(x))$ into the table.
- H_1-table: Initially empty, consists of tuples with form of (m, M). Once the adversary makes a (m, H_1) query that does not exist in H_1-table, it inserts $(m, \tilde{H}_1(m))$ into the table.
- P-table: Initially empty, consist of tuples with form of $(*, *, y, y')$. Once the adversary makes a (y, P) query which does not exist in P-table, it inserts $(*, *, \tilde{P}(y), y)$ into the table.
- P^{-1}-table: Initially empty, consist of tuples with form of $(*, *, y, y')$. Once the adversary makes a (y, P^{-1}) query which does not exist in P-table, it inserts $(*, *, y, \tilde{P}^{-1}(y))$ into the table.
- E-table: Initially empty, consist of tuples with form of (y, m, z, z'). Once the adversay makes a (y, m, z', E) query which does not exist in E-table, it inserts $(y, m, \tilde{E}_{y||m}(z'), z')$ into the table.
- E^{-1}-table: Initially empty, consist of tuples with form of (y, m, z, z'). Once the adversay makes a (y, m, z, E^{-1}) query which does not exist in E-table, it inserts $(y, m, z\tilde{E^{-1}}_{y||m}(z))$ into the table.

Note that at this point these tables are just keeping track of information relating to adversary's queries, and are completely hidden to the adversary. Next, same as above, we define a relation based on these tables. We will say a H_0 query $Q_k = (x, H_0)$ is in a table K, denoted $Q_k \in K$, if there is a 4-tuple in K such that the 1st element is equal to x. Analogously, for a P query we say $Q_k = (y, P) \in K$ if there is a 4-tuple in K such that the 4-th element is equal to y; for P^{-1} queries, we say $Q_k = (y, P^{-1}) \in K$ if there is a 4-tuple in K such that the 3rd element is equal to y. For a H_1-query, we say $Q_k = (m, H_1) \in K$ if there is a 2-tuple $T = (T_1, T_2)$in K such that $T_1 = m$; for an E-query, we say $Q_k = (y, m, z', E) \in K$ if there is a 4-tuple $T = (T_1, T_2, T_3, T_4) \in K$ such that $T_1 = y, T_2 = m, T_4 = z'$; for an E^{-1}-query, we say $Q_k = (y, m, z, E^{-1}) \in K$ if there is a 4-tuple $T = (T_1, T_2, T_3, T_4) \in K$ such that $T_1 = y, T_2 = m, T_3 = z$.

Then, we show a game that the system responds to the queries by using both tables and the real oracles without changing adversary's view.

Game 1. This game is identical to Game 0, except the way of maintaining the tables and responding to the queries. Specifically,

H_0-QUERY. Suppose (x, H_0) is the k-th query($k \in [1, q]$), the system responds as follows:

- Case 1. If $Q_k \in H_0$, then it responds with the corresponding output string;
- Case 2. If $Q_k \notin H_0$ but there is a tuple $T = (T_1, T_2, T_3, T_4)$ in P-table such that $T_4 = \mathsf{SKG}(x)$ and $T_2 \neq *$, then responds with T_2 and inserts (x, T_2, T_3, T_4) into H_0 table;
- Case 3. Otherwise, responds with $\tilde{H}_0(x)$ and inserts $(x, \tilde{H}_0(x), \mathsf{keygen}(\tilde{H}_0(x)), \mathsf{SKG}(x))$ and $(*, \tilde{H}_0(x), \mathsf{keygen}(\tilde{H}_0(x)), \mathsf{SKG}(x))$ into H_0-table and P-table, respectively.

P-QUERY. Suppose (y, P) be the k-th query, the system responds as follows:

- Case 1. If $Q_k \in H_0 \cup P \cup P^{-1}$, then respond with the third element of the 4-tuple corresponding to Q_k;
- Case 2. Otherwise, responds with $\tilde{P}(y)$, and inserts $(*, *, \tilde{P}(y), y)$ into both P and P^{-1} table.

P^{-1}-QUERY. Suppose (y, P) be the k-th query, the system responds as follows:

- Case 1. If $Q_k \in H_0 \cup P \cup P^{-1}$, then respond with the fourth element of the 4-tuple corresponding to Q_k;
- Case 2. Otherwise, responds with $\tilde{P}^{-1}(y)$, and inserts $(*, *, y, \tilde{P}^{-1}(y))$ into both P and P^{-1} table.

H_1-QUERY. Suppose (m, H_1) be the k-th query, the system responds as follows:

- Case 1. If $Q_k \in H_1$, then responds with the corresponding string;
- Case 2. Otherwise, the system responds with $\tilde{H}_1(m)$.

E-QUERY. Suppose (y, m, z', E) be the k-th query, the system firstly calls queries (m, H_1) and (y, P), then it responds as follows:

- If $Q_k \in E \cup E^{-1}$, then responds with 3rd element of the 4-tuple corresponding to Q_k;
- If $Q_k \notin E \cup E^{-1}$ and $\mathsf{SV}(y, m, z') = 0$, then the system responds with $\tilde{E}_{y||m}(z)$;
- If $Q_k \notin E \cup E^{-1}$ but $\mathsf{SV}(y, m, z') = 1$, then
  1. If there is a tuple $T = (T_1, T_2, T_3, T_4) \in H_0 \cup P$ such that $T_4 = y, T_2 \neq *$ then the system responds with $\mathsf{sign}(T_2, H_1(m))(H_1(m)$ is extracted from H_1-table);
  2. Otherwise, it responds with $\tilde{E}_{y,m}(z')$

E^{-1}-QUERY. Suppose (y, m, z, E^{-1}) be the k-th query, the system firstly calls queries (m, H_1) and (y, P), then it responds as follows:

- If $Q_k \in E \cup E^{-1}$, then responds with 4th element of the 4-tuple corresponding to Q_k;
- If $Q_k \notin E \cup E^{-1}$ and there is a tuple $T = (T_1, T_2, T_3, T_4) \in H_0 \cup P^{-1}$, such that $T_4 = y, T_1 \neq *, T_1' = m$, then the system tests $\mathsf{ver}(T_3, H_1(m), z) \overset{?}{=} 1(H_1(m)$ is extracted from H_1-table). If so, it responds with $\mathsf{SS}(T_1, m)$, else $\tilde{E^{-1}}_{y,m}(z)$;
- Otherwise, the system responds $\tilde{E^{-1}}_{y,m}(z)$.

Note that, in Game 1 the system keeps a longer table, and for each query, it firstly checks whether it can be responded by the tables or honest interfaces. If can, then it responds with them, else calls the real oracles. In fact, the system in Game 1 is a prototype for our simulator $\mathcal{S}$, and in the following games, we replace the responses that answered by the real oracles with random strings step by step. Moreover, the tuples stored in the table are consistent of the real oracles, hence each response in Game 1 is identical to the one in Game 0.

Game 2. This game is identical to Game 1, except for answering P queries. Assuming (y, P) to be the k-th query, the system responds as follows:

- Case 1: Suppose $Q_k \in H_0 \cup P \cup P^{-1}$, same as above.
- Case 2: If there is no such tuple in $\notin H_0 \cup P \cup P^{-1}$, but $\mathsf{SCC}_k = 1$, then respond with a random string $y' \leftarrow \mathcal{Y}$. The system inserts $(*, *, y', y)$ into the P-table.
- Case 3: If $Q_k \notin H_0 \cup P \cup P^{-1}$ and $\mathsf{SCC}_k = 0$, same as above.

Note that strong consistency check determines whether or not the adversary has the ability to learn $\tilde{P}(y)$ by making a query to SKG. If so, we must answer the P query using $\tilde{P}$; otherwise we can answer randomly.

Game 3. This game is identical to Game 2, except modifying P-query once more. More concretely, in case 2, if $Q_k \notin P \cup P^{-1}$ and $\mathsf{SCC}_k = 1$, then the system responds with a random public key $y' = \mathsf{keygen}(r)$, where $r \xleftarrow{\$} \mathcal{X}$ and inserts $(*, *, \mathsf{keygen}(r), y)$ into P-table.

Here, we just change the distribution of outputs in the case where we do not use $\tilde{P}$. Game 3 is indistinguishable from Game 2 by the pseudorandomness of the public key.

Game 4. This game is identical to Game 3, except the way of filling up the P-table. Concretely, in case 2, the system inserts the tuple $(*, r, \mathsf{keygen}(r), y)$ into the table.

Here, the only difference from Game 4 is that we record the random coins r used to sample the output of P into the P table. Since that r is not used with high probability , this is close to Game 4 from the adversary's perspective.

Game 5. This game is identical to Game 4, except that to answer P queries, where the system only uses the tables and honest interfaces. Assuming (y, P) to be the k-th query, then

- Case 1. $Q_k \in H_0 \cup P \cup P^{-1}$, same as above.
- Case 2. If $Q_k \notin H_0 \cup P \cup P^{-1}$ and $\mathsf{SCC}_k = 1$, then same as above;
- Case 3. If If $Q_k \notin H_0 \cup P \cup P^{-1}$ and $\mathsf{SCC}_k = 0 \cap \mathsf{WCC}_k = 1$, then it samples $r \leftarrow \mathcal{X}$, responds with $\mathsf{keygen}(r)$ and inserts $(*, r, \mathsf{keygen}(r), y)$ into P-table.

Game 5 and Game 4 are identical unless the adversary makes a query Q_k such that $Q_k = (y, P), \mathsf{SCC}_k = 0, \mathsf{WCC}_k = 1$ and $Q_k \notin P \cup P^{-1}$. This means that $y = \mathsf{SKG}(x) = \tilde{P}^{-1}(\mathsf{keygen}(\tilde{H}_0(x)))$ for some x, and that the adversary previously queried (x, SKG) or (x, m, SS), but not (x, H_0). Since $Q_k \notin P \cup P^{-1}$, the adversary has never made a query on $(\mathsf{keygen}(\tilde{H}_0(x)), P^{-1})$, as such a query would have resulted in y being added to the table for P^{-1}. Therefore, $\tilde{H}_0(x)$ and thus $\mathsf{keygen}(\tilde{H}_0(x))$ are independent of the adversary's view. In Game 5, the query would result in $(*, r, \mathsf{keygen}(\mathsf{r}), y)$ being added to the table for P, meanwhile in Game 4, the table records $(*, *, \mathsf{keygen}(\tilde{H}_0(x)), y)$. However, with high probability r is only used after (x, H_0) query, therefore, after the corresponding H_0 query, the P-table also records $(*, r, \mathsf{keygen}(\mathsf{r}), y)$. The only difference is that in Game 5, $r = \tilde{H}_0$, whereas in Game 6, r is a fresh random value. However, since the adversary has no knowledge of $\tilde{H}_0(x)$, the two games are indistinguishable.

Game 6. This game is identical to Game 5, except the way it answers P^{-1} queries, where the system *only* uses the tables to simulate if the strong consistency check pass. Assuming (y, P^{-1}) to be the k-th query, then,

- Case 1: If $Q_k \in H_0 \cup P \cup P^{-1}$, same as above;
- Case 2: If $Q_k \notin H_0 \cup P \cup P^{-1}$ and $\mathsf{SCC}_k = 1$, then it samples $x \leftarrow \mathcal{X}$, and responds with $\mathsf{SKG}(x)$ and inserts $(x, *, y, \mathsf{SKG}(x))$ into P^{-1}-table.
- Case 3. Otherwise, the system responds with $\tilde{P^{-1}}(y)$.

Note that $\mathsf{SCC}_k = 0$ implies that $Q_k \in H_0$, hence the case $Q_k \notin H_0 \cup P \cup P^{-1}$ and $\mathsf{SCC}_k = 0$ does not exist.

Due to the definition, Game 6 and Game 5 are identical unless there exists a query Q_k such that $Q_k = (y, P^{-1})$, $\mathsf{WCC}_k = 1$ and $Q_k \notin P \cup P^{-1}$. This means that, in the previous query, no x appears such that $y = \mathsf{keygen}(\tilde{H}_0(x))$, even in $\mathsf{SKG}, \mathsf{SS}$ queries, in other words, $\mathsf{SKG}(x)$ never appears and it is hidden from adversary's view. Thus we can replace it with a random value. Moreover, we can replace with $\mathsf{SKG(x')}$ where $x' \xleftarrow{\$} \mathcal{X}$, as long $\mathsf{keygen}(\tilde{H}_0(x'))$ never appears in the queries. We note that, as x' is randomly sampled and hidden from the adversary, thus except with negligible probability, $\mathsf{keygen}(\tilde{H}_0(x'))$ never appears.

Game 7. This game is identical to Game 6, except the way it answers P^{-1} queries, where the system *only* uses the tables to simulate, and not the true oracle $\tilde{P}^{-1}$. Assuming (y, P^{-1}) to be the k-th query, then,

- Case 1: If $Q_k \in H_0 \cup P \cup P^{-1}$, same as above;
- Case 2: If $Q_k \notin H_0 \cup P \cup P^{-1}$ and $\mathsf{SCC}_k = 1$, then same as above.
- Case 3: If $Q_k \notin H_0 \cup P \cup P^{-1}$ and $\mathsf{SCC}_k = 0, \mathsf{WCC}_k = 1$, then it samples $x \leftarrow \mathcal{X}$, and responds with $\mathsf{SKG}(x)$ and inserts $(x, *, y, \mathsf{SKG}(x))$ into P^{-1}-table.

Note that $\mathsf{WCC}_k = 0$ implies that $Q_k \in H_0$, hence the case $Q_k \notin H_0 \cup P \cup P^{-1}$ and $\mathsf{WCC}_k = 0$ does not exist.

Due to definition, Game 7 and Game 6 are identical unless there exists a query Q_k such that $Q_k = (y, P^{-1})$, $\mathsf{SCC}_k = 0, \mathsf{WCC}_k = 1$ and $Q_k \notin P \cup P^{-1}$. This means, there exist queries as (x, SKG) or $(x, m\mathsf{SS})$ such that $\mathsf{keygen}(\tilde{H}_0(x)) = y$. Moreover, by Game 5, we also note that query $(\mathsf{SKG}(x), P)$ would leak nothing about $\tilde{H}_0(x)$ and $\mathsf{keygen}(\tilde{H}_0(x))$. Hence, $\tilde{H}_0(x)$ and $\mathsf{keygen}(\tilde{H}_0(x))$ are hidden from the adversary's view. And if such an event happens, it means that the adversary is able to predict the public key for an unknown secret key. As our standard-model signature scheme has psedorandom public keys, this is impossible except with negligible probability.

Game 8. This game is identical to Game 7, except that to answer H_0 queries, where the system only uses the tables and honest interfaces. Assuming (x, H_0) is the k-th query, then

- Case 1. If $Q_k \in H_0$, same as above;
- Case 2. If $Q_k \notin H_0$ but there is a tuple $T = (T_1, T_2, T_3, T_4)$ in P-table such that $T_4 = \mathsf{SKG}(x)$ and $T_2 \neq *$, then same as above;
- Case 3. Otherwise, the system samples $r \leftarrow \mathcal{X}$, responds with r and inserts $(x, r, \mathsf{keygen}(r), \mathsf{SKG}(x))$ and $(*, r, \mathsf{keygen}(r), \mathsf{SKG}(x))$ into the H_0-table and P-table, respectively.

Recalling the H_0-table in Game 1, we immediately observe that, the only case that the system calls $\tilde{H}_0(x)$ is when the adversary knows nothing about $\tilde{H}_0(x)$ and $\mathsf{keygen}(\tilde{H}_0)(x)$, although the adversary might know $\mathsf{SKG}(x)$ and $(\mathsf{SKG}(x), P)$. Therefore, from the adversary's view, $\tilde{H}_0(x)$ is uniformly distributed in $\mathcal{X}$, and it is equivalent to randomly pick $r \leftarrow \mathcal{X}$ and implicitly set $r = (x, H_0)$ and $P^{-1}(\mathsf{keygen}(r)) = \mathsf{SKG}(x)$.

Game 9. This game is identical to Game 8, except that to answer H_1 query, where the system only uses the tables, not the true oracles. Assuming (m, H_1) is the k-th query, then

H_1-QUERY. Suppose (m, H_1) be the k-th query, the system responds as follows:

– Case 1. If $Q_k \in H_1$, then same as above;
– Case 2. Otherwise, the system randomly picks $M \leftarrow \mathcal{M}$, responds with M and inserts (m, M) into H_1-table.

Due to definition, we note that the only case that the system calls $\tilde{H_1}(m)$ is when the adversary knows nothing about $\tilde{H_1}(m)$, although the adversary might knows $\mathsf{SS}(x, m)$ for some secret key x. However, the adversary has not called $(\mathsf{SKG}(x), m, \mathsf{SS}(x, m), E)$ yet, otherwise (m, H_1) would be called automatically. Hence, $\tilde{H_1}(m)$ and $\mathsf{sign}(sk, \tilde{H_1}(m))$ are hidden from adversary's view and we can replace it with a random value.

Game 10. This game is identical to Game 9, expect that to answer E queries, where the system only uses the tables and honest interfaces. Assuming (y, m, z', E) is the k-th query, the system firstly calls queries (m, H_1) and (y, P), then it responds as follows:

– If $Q_k \in E \cup E^{-1}$, then responds with 3rd element of the 4-tuple corresponding to Q_k;
– If $Q_k \notin E \cup E^{-1}$ and $\mathsf{SV}(y, m, z') = 0$, then the system responds with $z \leftarrow \mathcal{Z}$,
– If $Q_k \notin E \cup E^{-1}$ but $\mathsf{SV}(y, m, z') = 1$, then the system responds with $\mathsf{sign}(T_2, H_1(m))$.

Game 11. This game is identical to Game 10, expect that to answer E^{-1} queries, where the system only uses the tables and honest interfaces. Assuming (y, m, z, E) is the k-th query, the system firstly calls queries (m, H_1) and (y, P), then it responds as follows:

– If $Q_k \in E \cup E^{-1}$, then same as above;
– If $Q_k \notin E \cup E^{-1}$ and there is a tuple $T = (T_1, T_2, T_3, T_4) \in H_0 \cup P^{-1}$, such that $T_4 = y, T_1 \neq *$, then the system tests $\mathsf{ver}(T_3, H_1(m), z) \overset{?}{=} 1$. If so, it responds with $\mathsf{SS}(T_1, m)$, else a random $z' \in \mathcal{Z}$;
– Otherwise, the system responds else a random $z' \in \mathcal{Z}$.

Game 12. In Game 11, the queries to H_0, H_1, P, P^{-1}, E and E^{-1} are answered by the tables which're maintained by the system and by making queries to $\mathsf{SKG}, \mathsf{SS}, \mathsf{SV}$. The system never makes queries directly to $\tilde{H}_0, \tilde{H}_1, \tilde{P}, \tilde{P}^{-1}, \tilde{E}, \tilde{E^{-1}}$; these oracles are *only* used to answer the $\mathsf{SKG}, \mathsf{SS}, \mathsf{SV}$ queries (either generated by the adversary or by the system's response to $H_0, H_1, P, P^{-1}, E, E^{-1}$ queries). At this point, it is straightforward to show that we can replace $\mathsf{SKG}, \mathsf{SS}$ and SV with the ideal versions from Definition 9, resulting in Game 12.

We note that in Game 12, the system is efficient, and it responds to the adversarial interfaces just by keeping several tables and accessing the honest interfaces. Thus, we can build a simulator that responds to $H_0, H_1, P, P^{-1}, E, E^{-1}$ queries exactly as the system does in Game 12. The result is that Game 12 corresponds to the ideal world. Since Game 0 is the real world, it suffices to prove that any adjacent games are indistinguishable. Next we will give the description of our simulator and prove the indistinguishability between each adjacent games.

Simulator. Let $\Pi_{\mathsf{Sig}} = (\mathsf{SKG}, \mathsf{SS}, \mathsf{SV})$ be an ideal signature scheme, associated with secret key space $\mathcal{X}$, public key space $\mathcal{Y}$, message space $\mathcal{M}$ and signature space $\mathcal{Z}$, where 1) $|\mathcal{X}| \leq |\mathcal{Y}|$; 2) $|\mathcal{M}| \leq |\mathcal{Z}|$. The simulator $\mathcal{S}$ responds to the queries as follows:

H_0-QUERY. Suppose (x, H_0) is the k-th query($k \in [1, q]$), the system responds as follows:

- Case 1. If $Q_k \in H_0$, then it responds with the corresponding output string;
- Case 2. If $Q_k \notin H_0$ but there is a tuple $T = (T_1, T_2, T_3, T_4)$ in P-table such that $T_4 = \mathsf{SKG}(x)$ and $T_2 \neq *$, then responds with T_2 and inserts (x, T_2, T_3, T_4) into H_0 table;
- Case 3. Otherwise, the system samples $r \leftarrow \mathcal{X}$, responds with r and inserts $(x, r, \mathsf{keygen}(r), \mathsf{SKG}(x))$ and $(*, r, \mathsf{keygen}(r), \mathsf{SKG}(x))$ into the H_0-table and P-table, respectively.

H_1-QUERY. Suppose (m, H_1) be the k-th query, the system responds as follows:

- Case 1. If $Q_k \in H_1$, then responds with the corresponding string;
- Case 2. Otherwise, the system randomly picks $M \leftarrow \mathcal{M}$, responds with M and inserts (m, M) into H_1-table.

P-QUERY. Suppose (y, P) be the k-th query, the system responds as follows:

- Case 1. If $Q_k \in H_0 \cup P \cup P^{-1}$, then respond with the third element of the 4-tuple corresponding to Q_k;
- Case 2.If $Q_k \notin H_0 \cup P \cup P^{-1}$ and $\mathsf{WCC}_k = 1$, then it samples $r \leftarrow \mathcal{X}$, responds with $\mathsf{keygen}(r)$ and inserts $(*, r, \mathsf{keygen}(r), y)$ into P-table.

P^{-1}-QUERY. Suppose (y, P) be the k-th query, the system responds as follows:

- Case 1. If $Q_k \in H_0 \cup P \cup P^{-1}$, then respond with the fourth element of the 4-tuple corresponding to Q_k;
- Case 2. If $Q_k \notin H_0 \cup P \cup P^{-1}$ and $\mathsf{WCC}_k = 1$, then it samples $x \leftarrow \mathcal{X}$, and responds with $\mathsf{SKG}(x)$ and inserts $(x, *, y, \mathsf{SKG}(x))$ into P^{-1}-table.

E-QUERY. Suppose (y, m, z', E) be the k-th query, the system firstly calls queries (m, H_1) and (y, P), then it responds as follows:

- If $Q_k \in E \cup E^{-1}$, then responds with 3rd element of the 4-tuple corresponding to Q_k;
- If $Q_k \notin E \cup E^{-1}$ and $\mathsf{SV}(y, m, z') = 0$, then the system responds with $z \leftarrow \mathcal{Z}$,
- If $Q_k \notin E \cup E^{-1}$ but $\mathsf{SV}(y, m, z') = 1$, then the system responds with $\mathsf{sign}(T_2, H_1(m))$.

E^{-1}-QUERY. Suppose (y, m, z, E^{-1}) be the k-th query, the system firstly calls queries (m, H_1) and (y, P), then it responds as follows:

- If $Q_k \in E \cup E^{-1}$, then responds with 4th element of the 4-tuple corresponding to Q_k;
- If $Q_k \notin E \cup E^{-1}$ and there is a tuple $T = (T_1, T_2, T_3, T_4) \in H_0 \cup P^{-1}$, such that $T_4 = y, T_1 \neq *$, then the system tests $\mathsf{ver}(T_3, H_1(m), z) \overset{?}{=} 1$. If so, it responds with $\mathsf{SS}(T_1, m)$, else a random $z' \in \mathcal{Z}$;
- Otherwise, the system responds else a random $z' \in \mathcal{Z}$.

Note that $\mathcal{S}$ works exactly the same as the system in Game 12. Moreover, the distribution of the honest interfaces in Game 12 is also identical to the ones in the ideal game, which means, for any adversary, the advantage in both Game 12 and Ideal Game is identical. Therefore, it suffices to prove that any adjacent games are indistinguishable.

Lemma 14. *Let Z be a variable such that $Z = \mathsf{sign}(r, m)$, where $r \leftarrow \mathcal{X}, m \leftarrow \mathcal{M}$, then for any $z \in \mathcal{Z}$,* $\Pr[Z = z] \leq \sqrt{\epsilon_2}$.

Proof. Assuming there exists z^* such that $\Pr[Z = z^*] > \sqrt{\epsilon_2}$, then it trivially breaks the RMA security by randomly samples m and r, and hands in its forgery as $\mathsf{sign}(r, m)$.

Claim. Game 0 $\approx$ Game 1.

Proof. It's trivial to note that Pr[Game 0] = Pr[Game 1], as in Game 1, the system only changes the way of maintaining the tables and the terms stored in each tuple is identical to the responses of real oracles.

Claim. Game 1 $\approx$ Game 2.

Proof. Firstly, we note that, in either Game 1 or Game 2, the adversary's view on the honest interfaces is consistent, hence it suffices to prove that, with high probability, the view on the adversarial interfaces also does not change. According to the definition, the only difference between Game 1 and Game 2 is when Case 3 occurs. Suppose (y, P) is the k-th query, the adversary's view on the adversarial interfaces are as follows:

View on H_0. By definition, $\tilde{H}_0$ is a random oracle such that 1) the response is consistent; 2) the response to a fresh input is uniformly random.

Thus, the adversary's view on H_0 is consistent in both games, if the responses of H_0 queries satisfy the two properties above. When case 3 occurs, the responses of H_0 queries do not change as it always responds with $\tilde{H}_0$.

View on P. By definition, $\tilde{P}$ is a random permutation oracle such that 1) the response is consistent; 2) the response to a fresh input is uniformly random; 3) the response is injective.

Thus, the adversary's view on P is consistent in both games, if the responses of P queries satisfy the three properties above. When case 3 occurs, after Q_k, the system sets $P(\mathsf{y}) = \mathsf{r}$. As r is randomly chosen, property 1, 2 hold trivially, and property 3 holds if there is no collision of P queries on r.

View on P^{-1}. By definition $\widetilde{P^{-1}}$ is the inverse of a random permuatation oracle such that 1) the response is consistent; 2) the response is the inverse of P; 3) the response is injective; 4) $P^{-1}(\mathsf{keygen}(H_0(\mathsf{x}))) = \mathsf{SKG}(\mathsf{x})$.

Thus, the adversary's view on P^{-1} is consistent in both games, if the responses of P queries satisfy the four properties above. When case 3 occurs, after Q_k, the system sets $P^{-1}(r) = y$, which implies property 2. Moreover, if r never appears in the previous queries, and $\tilde{P}(\mathsf{y}), \widetilde{P^{-1}}(\mathsf{r})$ never appear during the queries, then property 1 and 3 also hold. For property 4, And easy to note that property 4 holds if the adversary cannot output x^* such that $\mathsf{keygen}(\tilde{H}_0(x^*)) = \tilde{P}(y)$ or $\mathsf{keygen}(\tilde{H}_0(x^*)) = \mathsf{r}$. Applying Lemma 12, this bad event is bounded by $2q\sqrt{2\epsilon_1 + \frac{1}{|\mathcal{X}|}}$.

View on H_1. By definition, $\tilde{H}_1$ is a random oracle such that 1) the response is consistent; 2) the response to a fresh input is uniformly random.

Thus, the adversary's view on H_1 is consistent in both games, if the responses of H_1 queries satisfy the two properties above. When case 3 occurs, the responses of H_1 queries do not change as it always responds with $\tilde{H}_1$.

View on E. Suppose $(\mathsf{y}, \mathsf{m}, \mathsf{z}, E)$ to be an E query, then by definition, the view on E would satisfy the following:

- 1) the response is consistent;
- 2) the response is injective;

- 3) if $\mathsf{SV}(\mathsf{y},\mathsf{m},\mathsf{z}) = 1$, then $\mathsf{ver}(P(\mathsf{m}), H_1(\mathsf{m}), E_{\mathsf{y},\mathsf{m}}(\mathsf{z})) = 1$.
- 4) if $\mathsf{SV}(\mathsf{y},\mathsf{m},\mathsf{z}) = 0$, then $E_{\mathsf{y},\mathsf{m}}(\mathsf{z})$ is uniformly random and $\mathsf{ver}(P(\mathsf{m}), H_1(\mathsf{m}), E_{\mathsf{y},\mathsf{m}}(\mathsf{z})) = 0$;

Thus, the adversary's view on E is consistent in both games, if the responses of E queries satisfy the four properties above. When case 3 occurs, we note that the responses of the queries are consistent, which means property 1 and 2 trivially hold. For property 4, as $\Pi_{\mathsf{SM-Sig}}$ is unique signature scheme; if $\mathsf{SV}(\mathsf{y},\mathsf{m},\mathsf{z}) = 0$, then

$$\Pr[\mathsf{ver}(\mathsf{r}, H_1(\mathsf{m}), E_{\mathsf{y},\mathsf{m}}(\mathsf{z})) = 1] \leq \frac{1}{|\mathcal{Z}|}.$$

However, we note that, in game 2, $\tilde{P}(\mathsf{y})$ to r, then $\mathsf{ver}(P(\mathsf{m}), H_1(\mathsf{m}), E_{\mathsf{y},\mathsf{m}}(\mathsf{z})) = 1$ fails immediately, hence we need bound the probability that $\mathsf{SV}(\mathsf{y},\mathsf{m},\mathsf{z}) = 1$. In fact, as $\mathsf{SCC}_k = 1$, the adversary knows nothing of x such that $\mathsf{SKG}(\mathsf{x}) = y$, hence if $x, \tilde{H}_0(x)$ and $\mathsf{keygen}(\tilde{H}_0(x))$ never appears, then property 3 holds. Applying lemma 12, we can bound it by $\frac{q}{|\mathcal{X}|} + q\sqrt{2\epsilon_1 + \frac{1}{|\mathcal{X}|}}$.

View on E^{-1} Suppose $(\mathsf{y},\mathsf{m},\mathsf{z},E^{-1})$ to be an E query, then by definition, the view on E would satisfy the following:

- 1) the response is consistent;
- 2) the response is injective;
- 3) if $\mathsf{ver}(P(\mathsf{m}), H_1(\mathsf{m}), z) = 1$, then $\mathsf{SV}(\mathsf{y},\mathsf{m},E^{-1}_{\mathsf{y},\mathsf{m}}(z)) = 1$.
- 4) if $\mathsf{ver}(P(\mathsf{m}), H_1(\mathsf{m}), z) = 0$, then $E^{-1}_{\mathsf{y},\mathsf{m}}(z)$ is uniformly random and $\mathsf{SV}(\mathsf{y},\mathsf{m},E^{-1}_{\mathsf{y},\mathsf{m}}(z)) = 0$.

Thus, the adversary's view on E^{-1} is consistent in both games, if the responses of E^{-1} queries satisfy the four properties above. When case 3 occurs, we note that the responses of the queries are consistent, which means property 1 and 2 trivially hold. For property 4, same as above, we have that if $\mathsf{ver}(\tilde{P}(\mathsf{m}), H_1(\mathsf{m}), z) = 0$, then $\Pr[\mathsf{ver}(\mathsf{r}, H_1(\mathsf{m}), z) = 1] \leq \frac{1}{|\mathcal{Z}|}$. However, if $\mathsf{ver}(\tilde{P}(\mathsf{m}), H_1(\mathsf{m}), z) = 1$, then property 3 breaks immediately, hene we need bound the probability of the bad event. In fact, as we already assume $\tilde{H}_0$(the sign key of $\Pi_{\mathsf{SM-Sig}}$) never appears, this bad event can be bounded by $q\epsilon_2$, the RMA-security.

Now, we bound the union of these bad events as:

$$\Pr[\mathsf{Bad}] \leq \frac{2q}{|\mathcal{Y}|} + 3q\sqrt{2\epsilon_1 + \frac{1}{|\mathcal{X}|}} + \frac{2}{|\mathcal{Z}|} + \frac{q}{|\mathcal{X}|} + q\epsilon_2.$$

which referring to

$$|\Pr[\text{Game 1}] - \Pr[\text{Game 2}]| \leq q \cdot \Pr[\mathsf{Bad}] \leq \mathsf{negl}(\lambda).$$

Claim. Game 2 $\approx$ Game 3.

Proof. This proof is trivial as we just change the distribution of outputs from random strings to random public key. Hence, due to the pseudorandomness of keygen, we have

$$|\Pr[\text{Game 2}] - \Pr[\text{Game 3}]| \leq q\epsilon_1 \leq \mathsf{negl}(\lambda).$$

Claim. Game 3 $\approx$ Game 4.

Proof. In Game 4, the system only changes the way of maintaining the tables, and if r is not used, then the view in both games is consistent. Thus,

$$|\Pr[\text{Game 3}] = \Pr[\text{Game 4}]| \leq |\Pr[\text{Game 1}] = \Pr[\text{Game 2}]|.$$

Claim. Game 4 $\approx$ Game 5.

Proof. Firstly, we note that, in either Game 4 or Game 5, the adversary's view on the honest interfaces is consistent, hence it suffices to prove that, with high probability, the view on the adversarial interfaces also does not change. According to the definition, the only difference between Game 4 and Game 5 is when Case 3 occurs. Suppose (y, P) is the k-th query, where $\mathsf{y} = \mathsf{SKG}(\mathsf{x})$ and x is known by the adversary, then its view on the adversarial interfaces are as follows:

View on H_0. By definition, $\tilde{H_0}$ is a random oracle such that 1) the response is consistent; 2) the response to a fresh input is uniformly random.

Thus, the adversary's view on H_0 is consistent in both games, if the responses of H_0 queries satisfy the two properties above. When case 3 occurs, the system implicitly sets $H_0(\mathsf{x}) = \mathsf{r}$, as r is randomly chosen, and (x, H_0) never appears in previous queries, property 1 and 2 holds trivially.

View on P. By definition, $\tilde{P}$ is a random permutation oracle such that 1) the response is consistent; 2) the response to a fresh input is uniformly random; 3) the response is injective.

Thus, the adversary's view on P is consistent in both games, if the responses of P queries satisfy the three properties above. When case 3 occurs, after Q_k, the system sets $P(\mathsf{y}) = \mathsf{keygen}(r), \mathsf{r} \leftarrow \mathcal{X}$. Due to the pseudorandomness of the public key, property 1, 2 and 3 preserves.

View on P^{-1}. By definition $\tilde{P^{-1}}$ is the inverse of a random permuatation oracle such that 1) the response is consistent; 2) the response is the inverse of P; 3) the response is injective; 4) $P^{-1}(\mathsf{keygen}(H_0(\mathsf{x}))) = \mathsf{SKG}(\mathsf{x})$.

Thus, the adversary's view on P^{-1} is consistent in both games, if the responses of P queries satisfy the four properties above. When case 3 occurs, after Q_k, the system sets $P^{-1}(\mathsf{keygen}(\mathsf{r})) = \mathsf{y}$, which implies property 2 and 4. If $\mathsf{keygen}(\mathsf{r})$ never appears in the previous queries, and $\tilde{P}(\mathsf{y})$, $\tilde{H_0}(\mathsf{x})$ and $\tilde{H_0}(\mathsf{x}^*)$ such that $\mathsf{keygen}(\mathsf{x}^*) = \mathsf{r}$ never appear, then property 1 and 3 also holds. By lemma 12, the probability of this event is bounded by $3q\sqrt{2\epsilon_1 + \frac{1}{|\mathcal{X}|} + \frac{q}{|\mathcal{X}|}}$.

View on H_1. Same as above.

View on E. Suppose $(\mathsf{y}, \mathsf{m}, \mathsf{z}, E)$ to be an E query, then by definition, the view on E would satisfy the following:

- 1) the response is consistent;
- 2) the response is injective;
- 3) if $\mathsf{SV}(\mathsf{y}, \mathsf{m}, \mathsf{z}) = 1$, then $\mathsf{ver}(P(\mathsf{m}), H_1(\mathsf{m}), E_{\mathsf{y},\mathsf{m}}(\mathsf{z})) = 1$.
- 4) if $\mathsf{SV}(\mathsf{y}, \mathsf{m}, \mathsf{z}) = 0$, then $E_{\mathsf{y},\mathsf{m}}(\mathsf{z})$ is uniformly random and $\mathsf{ver}(P(\mathsf{m}), H_1(\mathsf{m}), E_{\mathsf{y},\mathsf{m}}(\mathsf{z})) = 0$;

Thus, the adversary's view on E is consistent in both games, if the responses of E queries satisfy the four properties above. When case 3 occurs, the system sets $E_{\mathsf{y},\mathsf{m}}(\mathsf{z}) = \mathsf{sign}(\mathsf{r}, H_1(\mathsf{m}))$, which implies property 3. Moreover, if $\mathsf{sign}(\mathsf{r}, H_1(\mathsf{m}))$ never appears in the previous queries and $\mathsf{sign}(\tilde{H_0}\mathsf{x}, H_1(\mathsf{m}))$m then the response is consistent and injective. Applying Lemma 14, this bad event can be bounded by $2q\sqrt{\epsilon_2}$. For property 4, as $\Pi_{\mathsf{SM-Sig}}$ is unique signature scheme; if $\mathsf{SV}(\mathsf{y},\mathsf{m},\mathsf{z}) = 0$, then

$$\Pr[\mathsf{ver}(\mathsf{r}, H_1(\mathsf{m}), E_{\mathsf{y},\mathsf{m}}(\mathsf{z})) = 1] \leq \frac{1}{|\mathcal{Z}|}.$$

View on E^{-1} Suppose $(\mathsf{y},\mathsf{m},\mathsf{z},E^{-1})$ to be an E query, then by definition, the view on E would satisfy the following:

- 1) the response is consistent;
- 2) the response is injective;
- 3) if $\mathsf{ver}(P(\mathsf{m}), H_1(\mathsf{m}), z) = 1$, then $\mathsf{SV}(\mathsf{y},\mathsf{m},E^{-1}_{\mathsf{y},\mathsf{m}}(z)) = 1$.
- 4) if $\mathsf{ver}(P(\mathsf{m}), H_1(\mathsf{m}), z) = 0$, then $E^{-1}_{\mathsf{y},\mathsf{m}}(z)$ is uniformly random and $\mathsf{SV}(\mathsf{y},\mathsf{m},E^{-1}_{\mathsf{y},\mathsf{m}}(z)) = 0$.

Thus, the adversary's view on E^{-1} is consistent in both games, if the responses of E^{-1} queries satisfy the four properties above. When case 3 occurs, the system sets $E^{-1}_{\mathsf{y},\mathsf{m}}(\mathsf{sign}(\mathsf{r}, H_1(\mathsf{m}))) = \mathsf{z}$,which implies property 3. For property 1, 2, 4, the analysis is same as above.

Now, we can bound the union of these bad events as:

$$\Pr[\mathsf{Bad}] \leq q\epsilon_1 + 3q\sqrt{2\epsilon_1 + \frac{1}{|\mathcal{X}|}} + \frac{q}{|\mathcal{X}|} + \frac{1}{|\mathcal{Z}|} + q\sqrt{\epsilon_2} \leq \mathsf{negl}(\lambda)$$

which referring to

$$|\Pr[\text{Game 4}] - \Pr[\text{Game 5}]| \leq q \cdot \Pr[\mathsf{Bad}] \leq \mathsf{negl}(\lambda).$$

Claim. Game 5 $\approx$ Game 6.

Proof. Firstly, we note that, in either Game 5 or Game 6, the adversary's view on the honest interfaces is consistent, hence it suffices to prove that, with high probability, the view on the adversarial interfaces also does not change. According to the definition, the only difference between Game 4 and Game 5 is when Case 2 occurs. Suppose (y, P^{-1}) is the k-th query and $\mathsf{SCC}_k = 1$, in Game 5 the system sets $P^{-1}(\mathsf{y}) = \mathsf{SKG}(\mathsf{x})$ where $\mathsf{x} \overset{\$}{\leftarrow} \mathcal{X}$, we note that adversary might know T such that $\mathsf{keygen}(\mathsf{T}) = \mathsf{y}$. Then the adversary's view on the adversarial interfaces are as follows:

View on H_0. In either Game 5 or Game 6, the responses of H_0 queries do not change.

View on P. By definition, $\tilde{P}$ is a random permutation oracle such that 1) the response is consistent; 2) the response to a fresh input is uniformly random; 3) the response is injective.

Thus, the adversary's view on P is consistent in both games, if the responses of P queries satisfy the three properties above. We note that, in both games $\tilde{P}$ is hidden and P-queries are answered by other tables and the honest interfaces, hence except negligible collision, property 3 holds.

View on P^{-1}. By definition $\tilde{P^{-1}}$ is the inverse of a random permuatation oracle such that 1) the response is consistent; 2) the response is the inverse of P; 3) the response is injective; 4) $P^{-1}(\mathsf{keygen}(H_0(\mathsf{x}))) = \mathsf{SKG}(\mathsf{x})$.

Thus, the adversary's view on P^{-1} is consistent in both games, if the responses of P queries satisfy the four properties above. When case 2 occurs, after Q_k, the system sets

$$P^{-1}(\mathsf{y}) = \mathsf{SKG}(x) = \tilde{P^{-1}}(\mathsf{keygen}(\tilde{H}_0(\mathsf{x})))$$

If $\mathsf{keygen}(\tilde{H}_0(\mathsf{x}))$ never appears in the previous queries, then $\mathsf{SKG}(x)$ is uniformly fresh, which means property 1, 2, 4 hold. For property 3, we note that if no x^*, such that $\mathsf{SKG}(\mathsf{x}^*) = \mathsf{SKG}(\mathsf{x})$ or $\mathsf{keygen}(\tilde{H}_0(\mathsf{x}^*)) = \mathsf{T}$ appear, then the responses are injective. Applying Lemma 12, we can bound it by $3q\sqrt{2\epsilon_1 + \frac{1}{|\mathcal{X}|}}$.

View on H_1. Same as above.

View on E. Suppose $(\mathsf{y}, \mathsf{m}, \mathsf{z}, E)$ to be an E query, then by definition, the view on E would satisfy the following:

- 1) the response is consistent;
- 2) the response is injective;
- 3) if $\mathsf{SV}(\mathsf{y}, \mathsf{m}, \mathsf{z}) = 1$, then $\mathsf{ver}(P(\mathsf{m}), H_1(\mathsf{m}), E_{\mathsf{y},\mathsf{m}}(\mathsf{z})) = 1$.
- 4) if $\mathsf{SV}(\mathsf{y}, \mathsf{m}, \mathsf{z}) = 0$, then $E_{\mathsf{y},\mathsf{m}}(\mathsf{z})$ is uniformly random and $\mathsf{ver}(P(\mathsf{m}), H_1(\mathsf{m}), E_{\mathsf{y},\mathsf{m}}(\mathsf{z})) = 0$;

Thus, the adversary's view on E is consistent in both games, if the responses of E queries satisfy the four properties above. Note that, in both game, the responses of E queries are consistent, which means property 1 and 2 hold trivially. Moreover, we note that if no x^* such that $\mathsf{keygen}(\tilde{H}_0(x^*)) = y$ and $\mathsf{sign}(\tilde{H}_0(x^*))$ appear then, property 3 and 4 also hold.

View on E^{-1}. Suppose $(\mathsf{y}, \mathsf{m}, \mathsf{z}, E^{-1})$ to be an E query, then by definition, the view on E would satisfy the following:

- 1) the response is consistent;
- 2) the response is injective;
- 3) if $\mathsf{ver}(P(\mathsf{m}), H_1(\mathsf{m}), z) = 1$, then $\mathsf{SV}(\mathsf{y}, \mathsf{m}, E^{-1}_{\mathsf{y},\mathsf{m}}(z)) = 1$.
- 4) if $\mathsf{ver}(P(\mathsf{m}), H_1(\mathsf{m}), z) = 0$, then $E^{-1}_{\mathsf{y},\mathsf{m}}(z)$ is uniformly random and $\mathsf{SV}(\mathsf{y}, \mathsf{m}, E^{-1}_{\mathsf{y},\mathsf{m}}(z)) = 0$.

Thus, the adversary's view on E^{-1} is consistent in both games, if the responses of E^{-1} queries satisfy the four properties above. When case 2 occurs, the system implictely sets $E^{-1}_{\mathsf{y},\mathsf{m}}(\mathsf{sign}(T, H_1(\mathsf{m}))) = \mathsf{SS}(\mathsf{x}, \mathsf{m})$,which implies property 3 immediately. For property 1, 2, 4, the analysis is same as above.

Now, we can bound the union of these bad events as:

$$\Pr[\mathsf{Bad}] \le \frac{q}{|\mathcal{Y}|} + 3q\sqrt{2\epsilon_1 + \frac{1}{|\mathcal{X}|}} + q\sqrt{2\epsilon_1 + \frac{1}{|\mathcal{X}|}} + q\sqrt{\epsilon_2} \le \mathsf{negl}(\lambda),$$

which referring to

$$|\Pr[\text{Game } 5] - \Pr[\text{Game } 6]| \le q \cdot \Pr[\mathsf{Bad}] \le \mathsf{negl}(\lambda).$$

Claim. Game 6 $\approx$ Game 7.

Proof. Suppose $Q_k = (\mathsf{y}, P^{-1})$ is the k-th query and we denote event Bad as $\mathsf{SCC_k} = 0 \cap \mathsf{WCC_k} = 1$. Next, we bound the bad event.

Assuming Bad occurs, then there is a x known by the adversary such that $y = \mathsf{keygen}(\widetilde{H_0}(\mathsf{x}))$ and (x, H_0) never appears in previous $k-1$ queries. As $\tilde{H_0}(\mathsf{x})$ is unifromly distributed in $\mathcal{X}$, the probability that it can output a proper y is bounded by the pseudorandomness of the public key. By definition, it's trivial to note that

$$|\Pr[\text{Game 6}] - \Pr[\text{Game 7}]| \le q \Pr[\mathsf{Bad}] \le q\epsilon_1 \le \mathsf{negl}(\lambda).$$

Claim. Game 7 $\approx$ Game 8.

Proof. Firstly, we note that, in either Game 7 or Game 8, the adversary's view on the honest interfaces is consistent, hence it suffices to prove that, with high probability, the view on the adversarial interfaces also does not change. According to the definition, the only difference between Game 7 and Game 8 is when Case 3 occurs. Suppose (x, H_0) is the k-th query, in Game 8 the system sets $H_0(\mathsf{x}) = \mathsf{r}$ where $\mathsf{r} \xleftarrow{\$} \mathcal{X}$. While we note that adversary might know $\mathsf{SKG}(\mathsf{x})$ before Q_k, with the restriction that $\mathsf{SKG}(\mathsf{x})$ never appears in $P \cup P^{-1}$ in the previous queries. Then the adversary's view on the adversarial interfaces are as follows:

View on H_0. By definition, $\tilde{H_0}$ is a random oracle such that 1) the response is consistent; 2) the response to a fresh input is uniformly random.

Thus, the adversary's view on H_0 is consistent in both games, if the responses of H_0 queries satisfy the two properties above. When case 3 occurs, the system implicitly sets $H_0(\mathsf{x}) = \mathsf{r}$, as r is randomly chosen, and (x, H_0) never appears in previous queries, property 1 and 2 holds trivially.

View on P and P^{-1}. In both games, P and P^{-1} are answered by H_0 table and the honest interfaces. As the view on H_0 does not change, except with negligible collision, the view on P and P^{-1} preserves.

View on H_1. Same as above.

View on E and E^{-1}. Similar to the analysis above, we have that, if $\mathsf{r}, \mathsf{keygen}(r)$ and $\mathsf{sign}(\mathsf{r}, *)$ never appears in the previous queries and $\tilde{H_0}(\mathsf{x}), \mathsf{keygen}(\tilde{H_0}(\mathsf{x}))$ and $\mathsf{sign}(\tilde{H_0}(\mathsf{x}), *)$ never appears then the view on E and E^{-1} is consistent in both games. Applying Lemma 12 and 14, we can bound this bad event by $\frac{2q}{|\mathcal{X}|} + 2q\sqrt{2\epsilon_1 + \frac{1}{|\mathcal{X}|}} + 2q\sqrt{\epsilon_2}$.

Now, we can bound the union of these bad events as:

$$\Pr[\mathsf{Bad}] \le \frac{q}{|\mathcal{Y}|} + 2q\sqrt{2\epsilon_1 + \frac{1}{|\mathcal{X}|}} + \frac{2q}{|\mathcal{X}|} + 2q\sqrt{\epsilon_2} \le \mathsf{negl}(\lambda)$$

which referring to

$$|\Pr[\text{Game 7}] - \Pr[\text{Game 8}]| \le q \cdot \Pr[\mathsf{Bad}] \le \mathsf{negl}(\lambda).$$

Claim. Game 8 $\approx$ Game 9.

Proof. Firstly, we note that, in either Game 8 or Game 9, the adversary's view on the honest interfaces is consistent, hence it suffices to prove that, with high probability, the view on the adversarial interfaces also does not change. According to the definition, we note that in Game 9, the system replaces the responses of (m, H_1) with a random string M.

View on H_0, P, P^{-1}. It's trivial that, in either Game 8 or Game 9, the H_0, P, P^{-1} queries are responded in the same way, hence the view does not change.

View on H_1. By definition, $\tilde{H_1}$ is a random oracle such that 1) the response is consistent; 2) the response to a fresh input is uniformly random.

Thus, the adversary's view on H_1 is consistent in both games, if the responses of H_1 queries satisfy the two properties above. As M is randomly sampled, if M never appears in the previous queries, then property 1 and 2 holds.

View on E and E^{-1}. Similar to the analysis above, we note that if $\mathsf{M}, \mathsf{sign}(*, \mathsf{M})$ never appear in the previous queries and $\tilde{H_0}(\mathsf{m}), \mathsf{sign}(*, \tilde{H_0}(\mathsf{m}))$ never appear, then view on E and E^{-1} is consistent. In fact, $\Pi_{\mathsf{SM-Sig}}$ has unique signature, we can easily bound the bad event by $\frac{4q}{|\mathcal{M}|}$. Thus, we have

$$|\Pr[\text{Game } 8] - \Pr[\text{Game } 9]| \le \frac{q^2}{|\mathcal{M}|} \le \mathsf{negl}(\lambda).$$

Claim. Game 9 $\approx$ Game 10.

Proof. Firstly, we note that, in either Game 9 or Game 10, the adversary's view on the honest interfaces is consistent, hence it suffices to prove that, with high probability, the view on the adversarial interfaces also does not change. According to the definition, we note that in Game 10, the system replaces the responses, which fails in the check, with a random string z'.

View on H_0, H_1, P, P^{-1}. It's trivial that, in either Game 9 or Game 10, the H_0, H_1, P, P^{-1} queries are responded in the same way, hence the view does not change.

View on E. Suppose $(\mathsf{y}, \mathsf{m}, \mathsf{z}, E)$ to be an E query, then by definition, the view on E would satisfy the following:

- 1) the response is consistent;
- 2) the response is injective;
- 3) if $\mathsf{SV}(\mathsf{y}, \mathsf{m}, \mathsf{z}) = 1$, then $\mathsf{ver}(P(\mathsf{m}), H_1(\mathsf{m}), E_{\mathsf{y},\mathsf{m}}(\mathsf{z})) = 1$.
- 4) if $\mathsf{SV}(\mathsf{y}, \mathsf{m}, \mathsf{z}) = 0$, then $E_{\mathsf{y},\mathsf{m}}(\mathsf{z})$ is uniformly random and $\mathsf{ver}(P(\mathsf{m}), H_1(\mathsf{m}), E_{\mathsf{y},\mathsf{m}}(\mathsf{z})) = 0$;

Thus, the adversary's view on E is consistent in both games, if the responses of E queries satisfy the four properties above. Property 3 holds trivial and if z' never appears in the previous queries, property 1 holds. Moreover, if $\mathsf{SV}(\mathsf{y}, \mathsf{m}, \mathsf{z}) = 0$, then

$$\Pr[\mathsf{ver}(\mathsf{r}, H_1(\mathsf{m}), E_{\mathsf{y},\mathsf{m}}(\mathsf{z})) = 1] \le \frac{1}{|\mathcal{Z}|}.$$

Property 2 holds trivially if no collision occurs.

View on E^{-1} Suppose $(\mathsf{y}, \mathsf{m}, \mathsf{z}, E^{-1})$ to be an E query, then by definition, the view on E would satisfy the following:

- 1) the response is consistent;
- 2) the response is injective;
- 3) if $\mathsf{ver}(P(\mathsf{m}), H_1(\mathsf{m}), z) = 1$, then $\mathsf{SV}(\mathsf{y}, \mathsf{m}, E^{-1}_{\mathsf{y},\mathsf{m}}(z)) = 1$.
- 4) if $\mathsf{ver}(P(\mathsf{m}), H_1(\mathsf{m}), z) = 0$, then $E^{-1}_{\mathsf{y},\mathsf{m}}(z)$ is uniformly random and $\mathsf{SV}(\mathsf{y}, \mathsf{m}, E^{-1}_{\mathsf{y},\mathsf{m}}(z)) = 0$.

As in Game 10, the system only replaces the value when $\mathsf{SV}(\mathsf{y},\mathsf{m},\mathsf{z})=0$, hence if z' never appears in the previous queries and $\tilde{E}_{\mathsf{y},\mathsf{m}}(\mathsf{z})$ never appears, then the view on E^{-1} preserves.

Now, we can bound the union of these bad events as:

$$\Pr[\mathsf{Bad}] \le \frac{2q+1}{|\mathcal{Z}|} + \frac{q}{|\mathcal{Z}|} \le \mathsf{negl}(\lambda)$$

which referring to

$$|\Pr[\text{Game 9}] - \Pr[\text{Game 10}]| \le q \cdot \Pr[\mathsf{Bad}] \le \mathsf{negl}(\lambda).$$

Claim. Game 10 $\approx$ Game 11.

Proof. Firstly, we note that, in either Game 10 or Game 11, the adversary's view on the honest interfaces is consistent, hence it suffices to prove that, with high probability, the view on the adversarial interfaces also does not change. According to the definition, we note that in Game 11, the system replaces the responses, which fails in the check, with a random string.

View on H_0, H_1, P, P^{-1}. It's trivial that, in either Game 8 or Game 9, the H_0, H_1, P, P^{-1} queries are responded in the same way, hence the view does not change.

Moreover, in both games, $\tilde{H}_0, \tilde{P}, \tilde{P^{-1}}$ has been completely hidden, hence the probability that $\mathcal{A}$ outputs $(\mathsf{y},\mathsf{m},\mathsf{z})$ such that z is a valid signature under $\tilde{H}_0, \tilde{H}_1\tilde{P}, \tilde{P^{-1}}$ is bounded by the uniqueness of the signature scheme. Therefore, it's rest to show that, if the query $(\mathsf{y},\mathsf{m},\mathsf{z})$ fails in the check, it would not change the view on property 3, 4 whp. In other words, suppose$(\mathsf{y},\mathsf{m},\mathsf{z},E^{-1})$ is the k-th query, then z is not the valid signature under H_0, P, P^{-1}, whp.

It's trivial that if z fails the validity check would not change the view, else then the adversary is asked to output a valid signature without knowing both sign key. Thus, we have

$$|\Pr[\text{Game 10}] - \Pr[\text{Game 11}]| \le \frac{q}{|\mathcal{M}|} + q\epsilon_2 \le \mathsf{negl}(\lambda).$$

Claim. Game 11 $\approx$ Game 12.

Proof. Due to definition, we note that in Game 11, the system responds all of the adversarial interfaces just using tables and honest interfaces, it never directly calls the real oracles. Besides, $\tilde{H}_0, \tilde{H}_1$ are random oracles, $\tilde{P}, \tilde{P^{-1}}$ are random permutations and $\tilde{E}, \tilde{E^{-1}}$ are ideal cipher models, hence the only chance the adversary can differ Game 9 and Game 10 is it can outputs $\mathsf{x} \neq \mathsf{x}' \in \mathcal{X}$ such that $\mathsf{keygen}(\tilde{H}_0(\mathsf{x})) = \mathsf{keygen}(\tilde{H}_0(\mathsf{x}'))$, which is bounded by the pseudorandomness of public key. Hence

$$|\Pr[\text{Game 11}] - \Pr[\text{Game 12}]| \le q^2\sqrt{2\epsilon_1 + \frac{1}{|\mathcal{X}|}} \le \mathsf{negl}(\lambda).$$

Combining together, we complete the whole proof. □

References

1. E. Andreeva, A. Bogdanov, Y. Dodis, B. Mennink, and J. P. Steinberger. On the indifferentiability of key-alternating ciphers. In *Advances in Cryptology–CRYPTO 2013*, pages 531–550. Springer, 2013.
2. M. Barbosa and P. Farshim. *Indifferentiable Authenticated Encryption: 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 1923, 2018, Proceedings, Part I*, pages 187–220. 01 2018.

3. M. Bellare, A. Boldyreva, and A. O'Neill. Deterministic and efficiently searchable encryption. In A. Menezes, editor, *Advances in Cryptology - CRYPTO 2007*, pages 535–552, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.
4. M. Bellare and P. Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *Proceedings of the 1st ACM conference on Computer and communications security*, pages 62–73. ACM, 1993.
5. J. Black. The ideal-cipher model, revisited: An uninstantiable blockcipher-based hash function. In *International Workshop on Fast Software Encryption*, pages 328–340. Springer, 2006.
6. D. Bleichenbacher. Chosen ciphertext attacks against protocols based on the rsa encryption standard pkcs #1. In H. Krawczyk, editor, *Advances in Cryptology — CRYPTO '98*, pages 1–12, Berlin, Heidelberg, 1998. Springer Berlin Heidelberg.
7. D. Boneh, B. Lynn, and H. Shacham. Short signatures from the weil pairing. *Journal of Cryptology*, 17(4):297–319, Sep 2004.
8. R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *Proceedings of the 42Nd IEEE Symposium on Foundations of Computer Science*, FOCS '01, pages 136–, Washington, DC, USA, 2001. IEEE Computer Society.
9. R. Canetti, O. Goldreich, and S. Halevi. The random oracle methodology, revisited. *J. ACM*, 51(4):557–594, July 2004.
10. D. Cash, E. Kiltz, and V. Shoup. The twin diffie-hellman problem and applications. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 127–145. Springer, 2008.
11. J.-S. Coron, Y. Dodis, C. Malinaud, and P. Puniya. Merkle-damgård revisited: How to construct a hash function. In *Annual International Cryptology Conference*, pages 430–448. Springer, 2005.
12. J.-S. Coron, Y. Dodis, A. Mandal, and Y. Seurin. A domain extender for the ideal cipher. In *Theory of Cryptography Conference*, pages 273–289. Springer, 2010.
13. J.-S. Coron, T. Holenstein, R. Künzler, J. Patarin, Y. Seurin, and S. Tessaro. How to build an ideal cipher: The indifferentiability of the feistel construction. *Journal of cryptology*, 29(1):61–114, 2016.
14. W. Diffie and M. Hellman. New directions in cryptography. *IEEE transactions on Information Theory*, 22(6):644–654, 1976.
15. Y. Dodis, T. Ristenpart, and T. Shrimpton. Salvaging merkle-damgård for practical applications. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 371–388. Springer, 2009.
16. Y. Dodis, M. Stam, J. Steinberger, and T. Liu. Indifferentiability of confusion-diffusion networks. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 679–704. Springer, 2016.
17. P. Farshim and G. Procter. The related-key security of iterated even–mansour ciphers. In *International Workshop on Fast Software Encryption*, pages 342–363. Springer, 2015.
18. A. Fiat and A. Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *Advances in CryptologyCRYPTO86*, pages 186–194. Springer, 1986.
19. E. S. Freire, D. Hofheinz, E. Kiltz, and K. G. Paterson. Non-interactive key exchange. In *Public-Key Cryptography–PKC 2013*, pages 254–271. Springer, 2013.
20. E. Fujisaki and T. Okamoto. Secure integration of asymmetric and symmetric encryption schemes. *Journal of Cryptology*, 26(1):80–101, Jan 2013.
21. R. Impagliazzo and S. Rudich. Limits on the provable consequences of one-way permutations. In S. Goldwasser, editor, *Advances in Cryptology — CRYPTO' 88*, pages 8–26, New York, NY, 1990. Springer New York.
22. D. Jost and U. Maurer. Security definitions for hash functions: Combining uce and indifferentiability. In D. Catalano and R. De Prisco, editors, *Security and Cryptography for Networks*, pages 83–101, Cham, 2018. Springer International Publishing.
23. U. Maurer, R. Renner, and C. Holenstein. Indifferentiability, impossibility results on reductions, and applications to the random oracle methodology. In *Theory of cryptography conference*, pages 21–39. Springer, 2004.
24. A. Mittelbach. Salvaging indifferentiability in a multi-stage setting. In P. Q. Nguyen and E. Oswald, editors, *Advances in Cryptology – EUROCRYPT 2014*, pages 603–621, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg.
25. T. Ristenpart, H. Shacham, and T. Shrimpton. Careful with composition: Limitations of the indifferentiability framework. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 487–506. Springer, 2011.
26. R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.

27. P. Rogaway and T. Shrimpton. A provable-security treatment of the key-wrap problem. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 373–390. Springer, 2006.
28. C. E. Shannon. Communication theory of secrecy systems. *Bell system technical journal*, 28(4):656–715, 1949.
29. V. Shoup. Lower bounds for discrete logarithms and related problems. In W. Fumy, editor, *Advances in Cryptology — EUROCRYPT '97*, pages 256–266, Berlin, Heidelberg, 1997. Springer Berlin Heidelberg.
30. M. Tibouchi. Elligator squared: Uniform points on elliptic curves of prime order as uniform random strings. In N. Christin and R. Safavi-Naini, editors, *Financial Cryptography and Data Security*, pages 139–156, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg.
31. B. Zhu, K. Li, and H. Patterson. Avoiding the disk bottleneck in the data domain deduplication file system. In *Proceedings of the 6th USENIX Conference on File and Storage Technologies*, FAST'08, pages 18:1–18:14, Berkeley, CA, USA, 2008. USENIX Association.