Indifferentiability for Public Key Cryptosystems

Mark Zhandry[1] and Cong Zhang[2]

[1] Princeton University & NTT Research, 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 relatively standard public key assumptions. By being indifferentiable from an ideal object, our schemes automatically satisfy a wide range of security properties, including any property representable 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 [28]. 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, it is difficult to predict the various applications devised by software developers that may deviate from the existing provably secure uses.

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 yet been discovered.

Ideal Hash Functions: The Random Oracle Model. Our inspiration 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 standard security notions for hash functions. But it is also much stronger: it is correlation intractable [7], 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 often 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 [25], the generic group model [26], and more recently ideal *symmetric* key encryption [2]. However, no prior work considers idealized models for public key cryptosystems.

Ideal 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 proven to be ideal in the random oracle model using Maurer's indifferentiability framework [20], 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 function, 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 and the formal definition for ideal NIKE is given in section 3.1.

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 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 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

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

a PRF. However, Maurer, Renner, and Holenstein [20] 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 the stronger 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, Renner, and Holenstein 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: Impagliazzo and Rudich [17] 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 [7] 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[4]. 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 [22] 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 [21] 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 [22] or context-restricted indifferentiability [19].

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[5], but we want to somehow incorporate keygen so that it can provide the structure needed when we turn to construct SHAREDKEY. 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

[4] There are actually several exceptions, for instance, the length-extension attacks against various Merkle-Damgard-based hash functions, such as MD5.

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

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.

Attack 1. Even if we assume keygen was collision resistant, the scheme would still not be indifferentiable. Indeed, the attacker 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 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. One possibility is to assume keygen has pseudorandom public keys; that is, that keygen applied to uniformly random coins gives a pseudorandom output.

Attack 2. 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: $\mathsf{KEYGEN}(\mathsf{SK}) = H_1(\mathsf{keygen}(H_0(\mathsf{SK})))$. 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 inputs and outputs for a standard-model scheme must be processed by idealized objects; this is the only way that the simulator can obtain enough information 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 one public key $\mathsf{PK}_1 = H_1(\mathsf{keygen}(H_0(\mathsf{SK}_1)))$, as well as another secret key SK_2. 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 [17] 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 $\mathsf{SHAREDKEY}(\mathsf{PK}, \mathsf{SK}) = \mathsf{sharedkey}(P(\mathsf{PK}), H_0(\mathsf{SK}))$. Note that, defining $\mathsf{SHAREDKEY}$ in this way achieves the desired correctness guarantee, which follows simply from the correctness of $(\mathsf{keygen}, \mathsf{sharedkey})$.

Attack 3. 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 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 $P(\mathsf{PK}) = \mathsf{keygen}(H_0(\mathsf{SK}))$.

Attack 4. Many 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 in the ideal world will give a random string. Then the adversary makes queries to P, H_0 and computes $\mathsf{sharedkey}(P(\mathsf{PK}'), H_0(\mathsf{SK}))$ for 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 $\mathsf{PK}'_1, \ldots, \mathsf{PK}'_\ell$ and $\mathsf{SK}_1, \ldots, \mathsf{SK}_\ell$, 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 $r_1, \ldots, r_\ell, s_1, \ldots, s_\ell$ 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 is to apply one more hash function, this time to the output of $\mathsf{sharedkey}$: $\mathsf{SHAREDKEY}(\mathsf{PK}_1, \mathsf{SK}_2) = H_1(\mathsf{sharedkey}(P(\mathsf{PK}_1), H_0(\mathsf{SK}_2)))$. 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 $H_1(\mathsf{sharedkey}(r_i, s_j)) = k_{i,j}$.

Attack 5. This construction unfortunately is still insecure: the adversary first samples sk_1 and SK_2, then it queries $\mathsf{PK}_2 = \mathsf{KEYGEN}(\mathsf{SK}_2), \mathsf{pk}_2 = P(\mathsf{PK}_2)$, then calculates $k = \mathsf{sharedkey}(\mathsf{pk}_2, \mathsf{sk}_1)$ and queries $H_1(k)$. After that, the adversary calculates $\mathsf{pk}_1 = \mathsf{keygen}(\mathsf{sk}_1)$, and queries $\mathsf{PK}_1 = P^{-1}(\mathsf{pk}_1)$. Next it calls $k' = \mathsf{SHAREDKEY}(\mathsf{PK}_1, \mathsf{SK}_2)$ and finally tests $k' \stackrel{?}{=} H_1(k)$. In the real world, the test always passes, while to achieve indifferentiability, the simulator has to output a proper $H_1(k)$. Unfortunately, until the query $H_1(k)$, simulator knows nothing of $(\mathsf{sk}_1, \mathsf{SK}_2)$ (it only has $\mathsf{KEYGEN}(\mathsf{SK}_2)$), which means that it cannot program $H_1(k)$ to be k'. Therefore test fails with overwhelming probability in the ideal world. To get prevent this attack, we present our final construction:

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

where $\mathsf{PK}_2 = \mathsf{KEYGEN}(\mathsf{SK}_2) = P^{-1}(\mathsf{keygen}(H_0(\mathsf{SK}_2)))$[6]. How does this help? We note that, in the final construction, by including $\mathsf{PK}_1, \mathsf{PK}_2$ in the H_1 queries, we force the adversary to query $\mathsf{PK}_1 = P^{-1}(\mathsf{pk}_1)$ before the H_1 query. This allows the simulator to program P^{-1} in a way that allows it to correctly answer the later H_1 query. In particular, it samples SK_1 itself and responds to the P^{-1} query with $\mathsf{PK}_1 = \mathsf{KEYGEN}(\mathsf{SK}_1)$. Afterward, when then adversary makes the query $H_1(\{\mathsf{PK}_1, \mathsf{PK}_2\}, k)$, the simulator will respond with $\mathsf{SHAREDKEY}(\mathsf{PK}_2, \mathsf{SK}_1)$, which is always identical to $\mathsf{SHAREDKEY}(\mathsf{PK}_1, \mathsf{SK}_2)$.

Attack 6. Even with our final construction, we must be careful. Suppose that it was possible for the adversary to choose a public key pk^* such that it can guess the value of $\mathsf{sharedkey}(\mathsf{pk}^*, \mathsf{sk})$ for a random (hidden) sk, then there is *still* an attack. Namely, the adversary queries $\mathsf{PK}^* = P^{-1}(\mathsf{pk}^*)$ and $\mathsf{PK} = \mathsf{KEYGEN}(\mathsf{SK})$ for a random SK. It then guesses the value t_0^* of $\mathsf{sharedkey}(\mathsf{pk}^*, H_0(\mathsf{SK}))$, without ever actually querying H_0, then it randomly samples an additional string t_1^* such that the simulator fails to distinguish t_0^* from t_1^* with high probability. Finally, it flips a coin b, and queries $k = H_1(\{\mathsf{PK}^*, \mathsf{PK}\}, t_b^*)$; if $b = 0$ it checks that the result is equal to $\mathsf{SHAREDKEY}(\mathsf{PK}^*, \mathsf{SK})$ and else it checks that the result is *not* equal to $\mathsf{SHAREDKEY}(\mathsf{PK}^*, \mathsf{SK})$. In the real world, if $b = 0$, this check will pass as long as the guess t_0^* is correct, and if $b = 1$, it will pass as long as no collision occurs. On the other hand, in the ideal world, although the simulator can simulate $k = H_1(\{\mathsf{PK}^*, \mathsf{PK}\}, t_0^*)$ correctly without knowing SK (the technique in Attack 5), it doesn't know the query corresponds to t_0^* or t_1^*. As a result, the check would fail with a noticeable probability. Our solution is to add another security requirement for the NIKE scheme, which we call *entropic*

[6] Here, $\{\mathsf{PK}_1, \mathsf{PK}_2\}$ means the un-ordered set containing PK_1 and PK_2, so that $\{\mathsf{PK}_1, \mathsf{PK}_2\} = \{\mathsf{PK}_2, \mathsf{PK}_1\}$.

shared keys, insisting that for any pk^* of the adversary's choosing, the adversary *cannot* guess $\mathsf{sharedkey}(\mathsf{pk}^*, \mathsf{sk})$ except for negligible probability.

Attack 7. One last attack strategy: the attacker could first query $\mathsf{PK}_1 = \mathsf{KEYGEN}(\mathsf{SK}_1), \mathsf{PK}_2 = \mathsf{KEYGEN}(\mathsf{SK}_2), k = \mathsf{SHAREDKEY}(\mathsf{PK}_1, \mathsf{SK}_2)$. Then, it could query $r_1 = P(\mathsf{PK}_1), r_2 = P(\mathsf{PK}_2)$. Finally, it could treat r_1, r_2 as the messages in the standard-model NIKE protocol, and guess the shared key t for the protocol. Then it could query H_1 on t (and $\{\mathsf{PK}_1, \mathsf{PK}_2\}$). In the real world, the result $H_1(\{\mathsf{PK}_1, \mathsf{PK}_2\}, t)$ would be equal to k, so the simulator in the ideal world needs to be able to set $H_1(\{\mathsf{PK}_1, \mathsf{PK}_2\}, t) = k$. At this point, the simulator has $\mathsf{PK}_1, \mathsf{PK}_2, t$. But the simulator has no knowledge of SK_1 or SK_2. Therefore it has no way of guessing the correct input to $\mathsf{SHAREDKEY}$ to obtain k, as doing so requires recovering either SK_1 or SK_2 by inverting KEYGEN.

Of course, this attack requires the adversary to guess the shared key t from the public messages r_1, r_2 in the standard-model NIKE scheme, which should be impossible. One difficulty is that, in our construction, the adversary implicitly has access to a kind of verification oracle for the standard-model NIKE, which allows it to input r_1, r_2 as well as a guess k' for k, and learn if the guess was correct. One of the r_1, r_2 can even be chosen by the adversary. To see how such an oracle arises, imagine the adversary queried $\mathsf{KEYGEN}(sk_1)$ to get PK_1 and $P(\mathsf{PK}_1)$ to get r_1. Then for an r_2 of its choice, it queries $P^{-1}(r_2)$ to get PK_2. Next, it queries $\mathsf{SHAREDKEY}(\mathsf{PK}_2, \mathsf{SK}_1)$ to get K, and $H_1(\mathsf{PK}_2, \mathsf{PK}_1, k')$ to get K'. The correctness of our algorithms implies that $K' = K$ if and only if $k' = k$.

Thus, if our NIKE scheme is only secure against passive attacks, it might be vulnerable to this attack vector. Concretely, if the underlying standard-model NIKE is the isogeny-based scheme of [18], then the active attack of [16] can be mounted against our scheme. Instead, we will require a stronger notion of NIKE security for the standard-model scheme, which we call *semi-active unpredictable shared keys*. Here, we require that the shared key is unpredictable, even if the adversary is given the verification oracles as described above. Such NIKE can easily be constructed under standard assumptions.

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 step in the hybrid argument 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 only a single element. We note that indifferentiable ideal ciphers can be constructed from random oracles [10].

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 [8].

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.

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 [15]. 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[7].

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 ideal 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 in addition to the semi-active unpredictable shared keys, we also need pseudorandom public keys and entropic shared keys. 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.

We note that the entropic shared key requirement is satisfied by all constructions we are aware of and the semi-active unpredictable shared keys can

[7] 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

be achieved on cryptographic groups or bilinear maps [14], under doubly-strong CDH assumption. On the other hand, the requirement of pseudorandom public keys 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 (which we need in order to apply the ideal permutation P), since $q^k = |\mathbb{Z}_q^k|$ may be far from a power of 2.

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. The result that we attain our NIKE results under doubly-strong-CDH, whereas our signature scheme requires CDH in bilinear map groups. The full details can be found in [27].

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 $\boldsymbol{m}$.

Random Oracle Model (ROM). Random oracle model is an idealized model proposed by Bellare and Rogaway [4]. ROM formalizes a model (a theoretical

black box) which responds to any unique query with a truly random string, and if the query is repeated, the response would be consistent. More concretely, a random oracle model has a publicly accessible hash function $H : \{0,1\}^* \to \{0,1\}^n$ such that:

1. for any x, every bit of $H(x)$ is truly random;
2. for any $x \neq y$, $H(x)$ and $H(y)$ are independent.

Ideal Cipher Model (ICM). The ideal cipher model is another idealized model which is firstly proposed by Shannon [25] and then formalized by Black [5]. This model also responds to any unique query with a truly random string. While, instead of having a publicly accessible random function, ideal cipher model has a publicly accessible ideal cipher $E : \{0,1\}^k \times \{0,1\}^n \to \{0,1\}^n$. Specifically, E is an ideal cipher along with a k-bit key and n-bit input/output such that:

1. for any pair (k, x), every bit of $E(k, x)$ is truly random;
2. for any fix key k, $E(k, *)$ is a random permutation;
3. for any $k_1 \neq k_2$ and (x, y), $E(k_1, x)$ and $E(k_2, y)$ are independent.

Moreover, any adversary interacting with an ideal cipher model would be given access to both the cipher and its inverse. The following definitions in this part highly rely on [2] and we roughly use its text here.

Games. An n-adversary game $\mathcal{G}$ is a Turing machine, denoted as $\mathcal{G}^{\Sigma,\mathcal{A}_1,\ldots,\mathcal{A}_n}$, where Σ is a system and $\mathcal{A}_i$ are adversarial algorithms that can keep full local state but might only communicate with each other through $\mathcal{G}$. If we say a n-adversary game $\mathcal{G}_n$ is reducible to an m-adversary game $\mathcal{G}_m$, we mean that, for any $(\mathcal{A}_1, \ldots, \mathcal{A}_n)$, there are $(\mathcal{A}'_1, \ldots, \mathcal{A}'_m)$ such that for any system Σ we have that $\mathcal{G}_n^{\Sigma,\mathcal{A}_1,\ldots,\mathcal{A}_n} = \mathcal{G}_m^{\Sigma,\mathcal{A}'_1,\ldots,\mathcal{A}'_m}$. A game $\mathcal{G}$ is called a n-stage game [22] if $\mathcal{G}$ is an n-adversary game and it cannot be reducible to any m-adversary game, where $m < n$. In particular, for any single-stage game $\mathcal{G}_{\Sigma,\mathcal{A}}$, we can trivially rewrite it as $\bar{\mathcal{A}}^{\bar{\mathcal{G}}^{\Sigma}}$, where $\bar{\mathcal{G}}$ is an oracle machine and $\bar{\mathcal{A}}$ is an adversarial algorithm, and $\bar{\mathcal{A}}$ is compatible with this oracle machine $\bar{\mathcal{G}}$. Moreover, we say two games are equivalent if they are reducible in both directions.

Random Functions. Let $\mathcal{X}$ and $\mathcal{Y}$ be two finite sets, we denote $\mathcal{F}[\mathcal{X} \to \mathcal{Y}]$ to be the set of all functions that map from $\mathcal{X}$ to $\mathcal{Y}$. If we say a function $F : \mathcal{X} \to \mathcal{Y}$ is a random function, we mean that F is uniformly sampled from $\mathcal{F}[\mathcal{X} \to \mathcal{Y}]$. Moreover, if F grants oracle accesses to all parties (honest and adversarial) in a black-box manner, then we treat F as an idealized model.

Random Injections. Similarly, let $\mathcal{X}$ and $\mathcal{Y}$ be two sets such that $|\mathcal{X}| \leq |\mathcal{Y}|$, we define $\mathcal{I}[\mathcal{X} \to \mathcal{Y}]$ to be the set of all injections that map from $\mathcal{X}$ to $\mathcal{Y}$. If we say a function $F : \mathcal{X} \to \mathcal{Y}$ is a random injection, we mean that F is uniformly sampled from $\mathcal{I}[\mathcal{X} \to \mathcal{Y}]$.

Lazy samplers. Lazy samplers are algorithmic procedures, to simulate various ideal objects along with arbitrary domain and range, by lazily sampling function at each point. Those ideal objects include: random oracle model, ideal cipher model, random functions and random injections [24].

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) [11]. 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. It consists of two algorithms: $\mathsf{NIKE.keygen}$ and $\mathsf{NIKE.sharedkey}$ together with a shared key space $\mathcal{SHK}$.

- $\mathsf{NIKE.keygen}$: Given input a secret key sk, the algorithm outputs a public key pk;
- $\mathsf{NIKE.sharedkey}$ Given inputs a public key $\mathsf{pk_1}$ and a secret key $\mathsf{sk_2}$, the algorithm outputs a shared key $\mathsf{shk} \in \mathcal{SHK}$.

For correctness, we require that, for any two key pairs $(\mathsf{pk_1}, \mathsf{sk_1}), (\mathsf{pk_2}, \mathsf{sk_2})$, the system satisfies:

$$\mathsf{NIKE.sharedkey}(\mathsf{pk_1}, \mathsf{sk_2}) = \mathsf{NIKE.sharedkey}(\mathsf{pk_2}, \mathsf{sk_1}).$$

Public Key Encryption (PKE) [11]. A public-key encryption scheme consists of three algorithms: $\mathsf{PKE.keygen}, \mathsf{PKE.enc}, \mathsf{PKE.dec}$ together with a message space $\mathcal{M}$. Formally,

- $\mathsf{PKE.keygen}$ Given input a secret key sk, the algorithm outputs a public key pk;
- $\mathsf{PKE.enc}$ Given inputs a public key pk and $m \in \mathcal{M}$, the algorithm outputs a ciphertext $\mathsf{c} = \mathsf{PKE.enc}(\mathsf{pk}, \mathsf{m})$;
- $\mathsf{PKE.dec}$ Given inputs a secret key sk and a ciphertext c, the algorithm outputs either a plaintext m or $\perp$.

For correctness, we require that, for any key pair $(\mathsf{pk}, \mathsf{sk})(\mathsf{pk} = \mathsf{PKE.keygen}(\mathsf{sk}))$ 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 [23]. A digital signature scheme consists of three algorithms: $\mathsf{Sig.keygen}, \mathsf{Sig.sign}, \mathsf{Sig.ver}$ along with a message space $\mathcal{M}$. Formally,

- $\mathsf{Sig.keygen}$ Given input a sign key sk, the algorithm outputs a verification key vk;
- $\mathsf{Sig.sign}$ Given inputs a sign key sk and a message $\mathsf{m} \in \mathcal{M}$, the algorithm outputs a signature $\sigma = \mathsf{Sig.sign}(\mathsf{sk}, \mathsf{m})$;
- $\mathsf{Sig.ver}$ Given 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 $(\mathsf{sk}, \mathsf{vk})$ $(\mathsf{vk} = \mathsf{Sig.keygen}(\mathsf{sk}))$ 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 Indifferentiability

For indifferentiability, significant parts of the discussion in this section is based on [2]. In [20], Maurer, Renner and Holenstein (MRH) propose the indifferentiability framework, which formalizes a set of necessary and sufficient conditions for one system to securely be replaced with another one in a wide class of environments. This framework has been used to prove the structual soundness of a number of cryptographic primitives, which includes hash functions [8, 12], blockciphers [1, 10, 13], domain extenders [9] and authenticated encryption with associated data [2]. In the following, we first recall the definition of indifferentiability.

A random system $\Sigma := (\Sigma.\text{hon}, \Sigma.\text{adv})$ is accessible via two interfaces $\Sigma.\text{hon}$ and $\Sigma.\text{adv}$, where $\Sigma.\text{hon}$ provides a honest interface through which the system can be accessed by all parties and $\Sigma.\text{adv}$ models the adversarial access to the inner working part of Σ. Typically, a system implements either some ideal objects $\mathcal{F}$, or a construction $C^{\mathcal{F}'}$, which applies some underlying ideal objects $\mathcal{F}'$.

Definition 1. (**Indifferentiability** [20, 2].) *Let Σ_1 and Σ_2 be two systems and $\mathcal{S}$ be a simulator. The indifferentiability advantage of a differentiator $\mathcal{D}$ against (Σ_1, Σ_2) with respect to $\mathcal{S}$ is*

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

where games $\text{Real}_{\Sigma_1,\mathcal{D}}$ and $\text{Ideal}_{\Sigma_2,\mathcal{S},\mathcal{D}}$ are defined in Figure 1. We say Σ_1 is indifferentiable from Σ_2, if there exists an efficient simulator $\mathcal{S}$ such that for any probabilistic polynomial time differentiator $\mathcal{D}$, the advantage above is negligible. Moreover, we say Σ_1 is weakly indifferentiable from Σ_2, if for any probabilistic polynomial time differentiator $\mathcal{D}$, there exists an efficient simulator $\mathcal{S}_\mathcal{D}$ such that the advantage above is negligible.

$\underline{\text{Real}_{\Sigma_1,\mathcal{D}}:}$	$\underline{\text{HonestR}(X)}$	$\underline{\text{Ideal}_{\Sigma_2,\mathcal{S},\mathcal{D}}:}$	$\underline{\text{HonestI}(X)}$
$b \leftarrow \mathcal{D}^{\text{HonestR},\text{AdvR}}$	Return $\Sigma_1.\text{hon}(X)$.	$b \leftarrow \mathcal{D}^{\text{HonestI},\text{AdvI}}$	Return $\Sigma_2.\text{hon}(X)$.
Return b.	$\underline{\text{AdvR}(X)}$	Return b.	$\underline{\text{AdvI}(X)}$
	Return $\Sigma_1.\text{adv}(X)$.		Return $\mathcal{S}^{\Sigma_2.\text{hon}(\cdot)}(X)$.

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 also use the notations in [2] and consider the definition above to two systems with interfaces as:

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

where $\mathcal{F}_1$ and $\mathcal{F}_2$ are two ideal objects sampled from their distributions and $\text{C}^{\mathcal{F}_1}$ is a construction of $\mathcal{F}_2$ by calling $\mathcal{F}_1$.

Next, we recall composition theorem for indifferentiability. In [20], MRH give out the composition theorem for indifferentiability, and then Ristenpart, Shacham and Shrimpton (RSS) [22] propose a game-based version for the theorem.

Theorem 2. (**Indifferentiability Composition** [22, 2].) *Let* $\Sigma_1 := (\mathrm{C}^{\mathcal{F}_1}, \mathcal{F}_1)$ *be a system that is indifferentiable from* $\Sigma_2 := (\mathcal{F}_2, \mathcal{F}_2)$ *along with simulator* $\mathcal{S}$*. Let* $\mathcal{G}$ *be a single-stage game. Then for any adversary* $\mathcal{A}$*, there exists an adversary* $\mathcal{B}$ *and a differentiator* $\mathcal{D}$ *such that*

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

However, RSS prove that the composition theorem above does not extend to multi-stage games as the simulator has to keep the local state for consistency. While, Barbosa and Farshim [2] observe that if allowing some relaxations on the games, we could rewrite some multi-stage games as equivalent to single-stage games. Essentially, for an n-adversary game $\mathcal{G}_n^{C^{\mathcal{F}},\mathcal{A}_1,\ldots,\mathcal{A}_n}$, if only one adversary (say $\mathcal{A}_1$) can call the ideal objects $\mathcal{F}$ directly and the rest can only call $C^{\mathcal{F}}$, then $\mathcal{G}_n$ can be rewritten as a single-stage game, because the game $\mathcal{G}_n$ itself, of course, has access to $C^{\mathcal{F}}$. Then in [2], BF formalize this observation in the following theorem.

Theorem 3. (**Multi-stage Game Composition** [2].) *Let* $\Sigma_1 := (\mathrm{C}^{\mathcal{F}_1}, \mathcal{F}_1)$ *be a system that is indifferentiable from* $\Sigma_2 := (\mathcal{F}_2, \mathcal{F}_2)$ *along with simulator* $\mathcal{S}$*. Let* $\mathcal{G}$ *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* $\mathcal{F}_1$ *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*

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

Remark. Barbosa and Farshim [2] give a strong motivation for the relaxation 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 any single-stage game. As a result, it would be insufficient to prove a system is related-key attack secure as follows: 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 allowing the relaxation, the proof follows trivially, hence from a practical point of view(by adding this specific relaxation on games), 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 $\mathcal{X}, \mathcal{Y}, \mathcal{W}$ be three sets such that $|\mathcal{X}| \geq 2^{\omega(\log \lambda)}$, $|\mathcal{Y}| \geq 2^{\omega(\log \lambda)}, |\mathcal{W}| \geq 2^{\omega(\log \lambda)}$, $|\mathcal{X}| \leq |\mathcal{Y}|$ and $|\mathcal{X}| \times |\mathcal{Y}| \leq |\mathcal{W}|$. We denote $\mathcal{F}[\mathcal{X} \to \mathcal{Y}]$ as the set of all injections that map $\mathcal{X}$ to $\mathcal{Y}$ and $\mathcal{G}[\mathcal{X} \times \mathcal{Y} \to \mathcal{W}]$ as the set of the functions that map $\mathcal{X} \times \mathcal{Y}$ to $\mathcal{W}$. 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) $G(x_1, y_1) = G(x_2, y_2) \Rightarrow (x_1 = x_2 \wedge y_1 = y_2) \vee (y_1 = F(x_2) \wedge y_2 = F(x_1))$.*

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 $\mathcal{W}$, 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 an information-theoretic argument, an ideal NIKE achieves related-key attack security, leakage-resiliency and so forth. Next, we show how to 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 consists 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.

Note that, the random permutations typically operate on bit strings, which means $\mathcal{Y} = \{0,1\}^n$ for some natural number $n \geq \omega(\log \lambda)$. Moreover, the shared key space in the standard model NIKE $\mathcal{Z}$ and in our construction $\mathcal{W}$ might be not equivalent, because it's unnecessarily correct that $|\mathcal{X}| \times |\mathcal{Y}| \leq |\mathcal{Z}|$. And if not, then setting $\mathcal{W} = \mathcal{Z}$ would give a differentiator directly, by just checking whether $|\mathcal{X}| \times |\mathcal{Y}| \leq |\mathcal{W}|$.

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,

- $\mathsf{NIKE.KEYGEN}(\mathsf{SK})$: Given input SK, the algorithm runs $\mathsf{keygen}(H_0(\mathsf{SK}))$, and outputs the public key $\mathsf{PK} = P^{-1}(\mathsf{keygen}(H_0(\mathsf{SK})))$;

– $\mathsf{NIKE.SHAREDKEY}(\mathsf{PK}_1, \mathsf{SK}_2)$: Given inputs $(\mathsf{PK}_1, \mathsf{SK}_2)$, the algorithm computes $\mathsf{PK}_2 = \mathsf{NIKE.KEYGEN}(\mathsf{SK}_2)$ and $\mathsf{sharedkey}(P(\mathsf{PK}_1), H_0(\mathsf{SK}_2))$. If $\mathsf{PK}_1 \leq \mathsf{PK}_2$, then it outputs the shared key as

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

else, 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 first specify the security properties of the standard-model NIKE.

Property 1. Semi-active unpredictable shared key. We say the shared key, for a NIKE scheme, is semi-active unpredictable, if there is ϵ_1 such thatfor any PPT adversary $\mathcal{A}$, the advantage

$$\mathrm{Adv}_{\mathcal{A}} := \Pr[\mathcal{A}^{\mathcal{O}_1, \mathcal{O}_2}(\mathsf{pk}_1, \mathsf{pk}_2) = \mathsf{sharedkey}(\mathsf{pk}_1, \mathsf{sk}_2)] \leq \epsilon_1 = \mathsf{negl}(\lambda),$$

where $\mathsf{pk}_i = \mathsf{keygen}(\mathsf{sk}_i), \mathsf{sk}_i \leftarrow \mathcal{X}$ and $\mathcal{O}_i$ is a predicate oracle such that takes $(\mathsf{pk_i}, \mathsf{shk})$ as input and outputs a bit (the public key pk here might be malicious). Concretely, the oracle $\mathcal{O}_i$ outputs "1" iff $\mathsf{shk} = \mathsf{sharedkey}(\mathsf{sk_i}, \mathsf{pk})$. This is the standard security game for NIKE schemes against active adversary, except that we relax the notion on two pieces: 1) we only require unpredictability of the shared key, rather than indistinguishability from random; 2) the oracles take both public key pk and shared key shk as input and tell whether shk is a valid shared key, rather than taking the public key pk, and outputting the corresponding shared key shk.

The next two properties are mild additional security properties that are not usually required for NIKE schemes, but are achieved by most natural schemes. We require these properties in order to prove the ideal security of our construction.

Property 2. entropic shared keys. We say the shared key, for a NIKE scheme is entropic, if there is ϵ_2 s.t. for any PPT adversary $\mathcal{A}$, the advantage

$$\Pr[\mathsf{shk}^* = \mathsf{sharedkey}(\mathsf{pk}^*, \mathsf{sk}) : (\mathsf{pk}^*, \mathsf{shk}^*) \leftarrow \mathcal{A}, \mathsf{sk} \xleftarrow{\$} \mathcal{X}] \leq \epsilon_2 = \mathsf{negl}(\lambda),$$

Note that the entropic shared keys property tells us that if the adversary only knows one public key (even it's chosen by the adversary), it cannot predict the shared key if the other secret key is random and hidden. In other words, this property guarantees that there is no way to make the shared key have low min-entropy.

Property 3. pseudorandom public keys. We say the public key, for a NIKE scheme, is pseudorandom, if there is ϵ_3 s.t. for any PPT adversary $\mathcal{A}$, the advantage

$$\mathrm{Adv}_{\mathcal{A}} := |\Pr[\mathcal{A}(\mathsf{keygen}(\mathsf{sk}))] - \Pr[\mathcal{A}(R)]| \leq \epsilon_3 = \mathsf{negl}(\lambda),$$

where $\mathsf{sk} \leftarrow \mathcal{X}, R \leftarrow \mathcal{Y}$. We immediately observe that as $\mathcal{Y} = \{0,1\}^n$, our standard-model NIKE must have public keys that are pseudorandom bit strings. And we say a NIKE scheme is Good if it satisfies the three properties above.

Theorem 5. *(***(Indifferentiable NIKE).***)* Π_{NIKE} *is indifferentiable from an ideal NIKE if* $\Pi_{\mathsf{SM-NIKE}}$ *is* Good*. More precisely, there exists a simulator* $\mathcal{S}$ *such that for all* $(q_{H_0}, q_{H_0}, q_{H_0}, q_{H_0})$*-query PPT differentiator* $\mathcal{D}$ *with* $q_{H_0} + q_P + q_{P^{-1}} + q_{H_1} \leq q$*, we have*

$$\mathrm{Adv}^{\mathsf{indif}}_{\Pi_{\mathsf{NIKE}},\mathcal{S},\mathcal{D}} \leq (8q^2+9q)\sqrt{2\epsilon_3 + \frac{1}{|\mathcal{X}|}} + \frac{4q^2+5q}{|\mathcal{Y}|} + \frac{q^2}{|\mathcal{W}|} + 4q^2\epsilon_1 + (q^2+11q)\epsilon_2 + 9q\epsilon_3.$$

The simulator makes at most q *queries to its oracles.*

Proof Sketch. According to the definition of indifferentiability, we immediately observe that any PPT 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, P, P^{-1}, H_1). Therefore, we need to build an efficient simulator $\mathcal{S}$ that can simulate the four adversarial interfaces H_0, P, P^{-1} and H_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. In the following, we illustrate the description of our simulator (similar form as in [2]) in Figure 2 and then we give the high-level intuition of our proof strategy (here we only give the proof sketch and the intuition of the simulator, and please refer to [27] for full details).

We immediately observe that, our simulator makes at most q queries to $(\mathsf{NKG}, \mathsf{NSK})$, and it keeps four tables and the size of each table is at most q, referring to $\mathcal{S}$ is efficient. In the following, we present the intuitive idea that why $\mathcal{S}$ works. Note that, in the real game, H_0, H_1 are random oracles, P is a random permutation associated with its inverse P^{-1}. Hence, the responses of a proper simulator should follow the following rules:

1. The responses of H_0, H_1 are computational close to uniform distribution;
2. The responses of P, P^{-1} are computational close to a random permutation;
3. There do not exist $(\mathsf{PK}_1 \neq \mathsf{PK}_2), (\mathsf{pk}_1 \neq \mathsf{pk}_2)$ such that $P^{(}\mathsf{PK}_1) = P^{(}\mathsf{PK}_2)$ or $P^{-1}(\mathsf{pk}_1) = P^{-1}(\mathsf{pk}_2)$;
4. $\mathsf{NKG}(\mathsf{SK}) = P^{-1}(\mathsf{keygen}(H_0(\mathsf{SK})))$;
5. $\mathsf{NSK}(\mathsf{SK}_1, \mathsf{PK}_2) = \mathsf{NSK}(\mathsf{SK}_2, \mathsf{PK}_1) = H_1(\{\mathsf{PK}_1, \mathsf{PK}_2\}^8, \mathsf{sharedkey}(\mathsf{sk}_1, \mathsf{pk}_2))$.

Next, we illustrate why and how $\mathcal{S}$ achieves these five rules.

Rule 1. Easy to note that the response of any H_0 query ($H_0(\mathsf{SK})$) is well-formed. Roughly, $\mathcal{S}$ responds to it using T_{H_0}, T_P (the second term of the corresponding tuple) or with a random string $\mathsf{sk} \in \mathcal{X}$. Moreover, note that the second term in every tuple from table T_{H_0} or T_P is uniformly sampled, referring to $H_0(\mathsf{SK})$ is uniformly distributed. For H_1 query, say $H_1(\mathsf{PK}_1, \mathsf{PK}_2, \mathsf{shk})$, the simulator

[8] If $\mathsf{PK}_1 \leq \mathsf{PK}_2$, $\{\mathsf{PK}_1, \mathsf{PK}_2\} = (\mathsf{PK}_1, \mathsf{PK}_2)$, and else $\mathsf{PK}_1 \leq \mathsf{PK}_2 = (\mathsf{PK}_2, \mathsf{PK}_1)$.

Algo.$\mathcal{S}.H_0(\mathsf{SK})$:
if $\exists(\mathsf{SK}, \mathsf{sk}, \mathsf{pk}, \mathsf{PK}) \in T_{H_0}$,
  return sk;
if $\exists(*, \mathsf{sk}, \mathsf{pk}, \mathsf{PK}) \in T_P$ s.t. $\mathsf{PK} = \mathsf{NKG}(\mathsf{SK})$,
  return sk;
$\mathsf{sk} \twoheadleftarrow \mathcal{X}$,
$T_{H_0} \leftarrow T_{H_0} \cup (\mathsf{SK}, \mathsf{sk}, \mathsf{keygen}(\mathsf{sk}), \mathsf{NKG}(\mathsf{SK}))$,
  return sk.

Algo.$\mathcal{S}.P(\mathsf{PK})$
if $\exists(*, \mathsf{sk}, \mathsf{pk}, \mathsf{PK}) \in T_P$,
  return pk;
if $\exists(\mathsf{SK}, *, \mathsf{pk}, \mathsf{PK}) \in T_{P^{-1}}$,
  return pk;
if $\exists(\mathsf{SK}, \mathsf{sk}, \mathsf{pk}, \mathsf{PK}) \in T_{H_0}$,
  return pk;
$\mathsf{sk} \twoheadleftarrow \mathcal{X}$, $T_P \leftarrow T_P \cup (*, \mathsf{sk}, \mathsf{keygen}(\mathsf{sk}), \mathsf{PK})$,
  return $\mathsf{keygen}(\mathsf{sk})$.

Algo.$\mathcal{S}.P^{-1}(\mathsf{pk})$:
if $\exists(\mathsf{SK}, *, \mathsf{pk}, \mathsf{PK}) \in T_{P^{-1}}$, return PK;
if $\exists(*, \mathsf{sk}, \mathsf{pk}, \mathsf{PK}) \in T_P$, return PK;
if $\exists(\mathsf{SK}, \mathsf{sk}, \mathsf{pk}, \mathsf{PK}) \in T_{H_0}$, return PK;
$\mathsf{SK} \twoheadleftarrow \mathcal{X}$, $T_{P^{-1}} \leftarrow T_{P^{-1}} \cup (\mathsf{SK}, *, \mathsf{pk}, \mathsf{NKG}(\mathsf{SK}))$, return $\mathsf{NKG}(\mathsf{SK})$.

Algo.$\mathcal{S}.H_1(\mathsf{PK}_1, \mathsf{PK}_2, \mathsf{shk})$:
if $\exists(\mathsf{PK}_1, \mathsf{PK}_2, \mathsf{shk}, \mathsf{SHK}) \in T_{H_1}$, return SHK;
$\mathsf{w} \twoheadleftarrow \mathcal{W}$,
if $\mathsf{PK}_1 > \mathsf{PK}_2$, $T_{H_1} \leftarrow T_{H_1} \cup (\mathsf{PK}_1, \mathsf{PK}_2, \mathsf{shk}, \mathsf{w})$, return w;
else if $\exists(\mathsf{SK}_1, \mathsf{sk}_1, \mathsf{pk}_1, \mathsf{PK}_1) \in T_{H_0}, (\mathsf{SK}_2, \mathsf{sk}_2, \mathsf{pk}_2, \mathsf{PK}_2) \in T_{H_0}$,
  if $\mathsf{shk} = \mathsf{sharedkey}(\mathsf{sk}_1, \mathsf{pk}_2)$, return $\mathsf{NSK}(\mathsf{SK}_1, \mathsf{PK}_2)$;
  else $T_{H_1} \leftarrow T_{H_1} \cup (\mathsf{PK}_1, \mathsf{PK}_2, \mathsf{shk}, \mathsf{w})$, return w.
if $\exists(\mathsf{SK}_1, \mathsf{sk}_1, \mathsf{pk}_1, \mathsf{PK}_1) \in T_{H_0}$, $(*, \mathsf{sk}_2, \mathsf{pk}_2, \mathsf{PK}_2) \in T_P$,
  if $\mathsf{shk} = \mathsf{sharedkey}(\mathsf{sk}_1, \mathsf{pk}_2)$, return $\mathsf{NSK}(\mathsf{SK}_1, \mathsf{PK}_2)$;
  else $T_{H_1} \leftarrow T_{H_1} \cup (\mathsf{PK}_1, \mathsf{PK}_2, \mathsf{shk}, \mathsf{w})$, return w.
if $\exists(\mathsf{SK}_1, \mathsf{sk}_1, \mathsf{pk}_1, \mathsf{PK}_1) \in T_{H_0}$, $(\mathsf{SK}_2, *, \mathsf{pk}_2, \mathsf{PK}_2) \in T_{P^{-1}}$,
  if $\mathsf{shk} = \mathsf{sharedkey}(\mathsf{sk}_1, \mathsf{pk}_2)$, return $\mathsf{NSK}(\mathsf{SK}_1, \mathsf{PK}_2)$;
  else $T_{H_1} \leftarrow T_{H_1} \cup (\mathsf{PK}_1, \mathsf{PK}_2, \mathsf{shk}, \mathsf{w})$, return w.
if $\exists(\mathsf{SK}_2, \mathsf{sk}_2, \mathsf{pk}_2, \mathsf{PK}_2) \in T_{H_0}$, $(*, \mathsf{sk}_1, \mathsf{pk}_1, \mathsf{PK}_1) \in T_P$,
  if $\mathsf{shk} = \mathsf{sharedkey}(\mathsf{sk}_2, \mathsf{pk}_1)$, return $\mathsf{NSK}(\mathsf{SK}_2, \mathsf{PK}_1)$;
  else $T_{H_1} \leftarrow T_{H_1} \cup (\mathsf{PK}_1, \mathsf{PK}_2, \mathsf{shk}, \mathsf{w})$, return w.
if $\exists(\mathsf{SK}_2, \mathsf{sk}_2, \mathsf{pk}_2, \mathsf{PK}_2) \in T_{H_0}$, $(\mathsf{SK}_1, *, \mathsf{pk}_1, \mathsf{PK}_1) \in T_{P^{-1}}$,
  if $\mathsf{shk} = \mathsf{sharedkey}(\mathsf{sk}_2, \mathsf{pk}_1)$, return $\mathsf{NSK}(\mathsf{SK}_2, \mathsf{PK}_1)$;
  else, $T_{H_1} \leftarrow T_{H_1} \cup (\mathsf{PK}_1, \mathsf{PK}_2, \mathsf{shk}, \mathsf{w})$, return w.
if $\exists(\mathsf{SK}_1, *, \mathsf{pk}_1, \mathsf{PK}_1) \in T_{P^{-1}}$, $(*, \mathsf{sk}_2, \mathsf{pk}_2, \mathsf{PK}_2) \in T_P$,
  if $\mathsf{shk} = \mathsf{sharedkey}(\mathsf{sk}_2, \mathsf{pk}_1)$, return $\mathsf{NSK}(\mathsf{SK}_1, \mathsf{PK}_2)$;
  else, $T_{H_1} \leftarrow T_{H_1} \cup (\mathsf{PK}_1, \mathsf{PK}_2, \mathsf{shk}, \mathsf{w})$, return w.
if $\exists(\mathsf{SK}_2, *, \mathsf{pk}_2, \mathsf{PK}_2) \in T_{P^{-1}}$, $(*, \mathsf{sk}_1, \mathsf{pk}_1, \mathsf{PK}_1) \in T_P$,
  if $\mathsf{shk} = \mathsf{sharedkey}(\mathsf{sk}_1, \mathsf{pk}_2)$, return $\mathsf{NSK}(\mathsf{SK}_2, \mathsf{PK}_1)$;
  else, $T_{H_1} \leftarrow T_{H_1} \cup (\mathsf{PK}_1, \mathsf{PK}_2, \mathsf{shk}, \mathsf{w})$, return w.
$T_{H_1} \leftarrow T_{H_1} \cup (\mathsf{PK}_1, \mathsf{PK}_2, \mathsf{shk}, \mathsf{w})$, return w.

Fig. 2: Simulator for NIKE in terms of four sub-simulators associated with two oracles $(\mathsf{NKG}, \mathsf{NSK})$. These four sub-simulators share four tables $(T_{H_0}, T_P, T_{P^{-1}}, T_{H_1})$ as joint state (which are initialized empty). The commands, e.g."$\exists(\mathsf{SK}, \mathsf{sk}, \mathsf{pk}, \mathsf{PK}) \in T_{H_0}$", go through the table in some well-defined order.

responds to it with either $\mathsf{NSK}(\mathsf{SK}_1, \mathsf{PK}_2)$ (if the tests pass) or a random string $\mathsf{w} \in \mathcal{W}$. And we note that $(\mathsf{NKG}, \mathsf{NSK})$ is an ideal NIKE, which means that $\mathsf{NSK}(\mathsf{SK}_1, \mathsf{PK}_2)$ is uniformly distributed.

Rule 2. For P query, say $P(\mathsf{PK})$, the simulator responds to the query in four cases: 1) using T_P; 2) using $T_{P^{-1}}$; 3) using T_{H_0}; 4) randomly sampling a secret key and outputting the corresponding public key. Easy to note that, in case 1, 3, and 4, $\mathcal{S}$ always returns a random public key, and due to the pseudo-random public keys, we have that the response is computational close to uniform. For Case 2, note that if $\mathsf{PK} \in \mathsf{T}_{\mathsf{P}^{-1}}$, then we know that the adversary has made a query $P^{-1}(\mathsf{pk})$ previously. For that P^{-1} query, $\mathcal{S}$ samples $\mathsf{SK} \in \mathcal{X}$ and responds to it with $\mathsf{PK} = \mathsf{NKG}(\mathsf{SK})$. As $(\mathsf{NKG}, \mathsf{NSK})$ is an ideal NIKE, we have that PK is close to uniform in $\mathcal{Y}$ and the response of $P^{-1}(\mathsf{pk})$ is well distributed. As a result, when making a query $P(\mathsf{PK})$, pk is the proper answer. Moreover, P^{-1} is P's inverse, which means the responses of P^{-1} are also well-distributed.

Rule 3. This rule indicates that P and P^{-1} must be bijective. For $\mathcal{S}$, note that there are four bad cases that break the rule:

1. pk-collision: $\mathcal{A}$ makes a query $P(\mathsf{pk}^*)$ with response PK_1^*, and then it makes another query $P^{-1}(\mathsf{PK}_2^*)$ with response pk^*;
2. PK-collision: $\mathcal{A}$ makes a query $P(\mathsf{PK}^*)$ with response pk_1^*, and then it makes another query $P(\mathsf{pk}_2^*)$ with response $\mathsf{NKG}(\mathsf{SK}^*) = \mathsf{PK}^*$;
3. Guessed-$H_0(\mathsf{SK})$-on-pk^*: adversary makes a query $P^{-1}(\mathsf{PK}^*)$ with response $\mathsf{keygen}(\mathsf{sk}^*)$, and $\mathcal{A}$ also makes a query $H_0(\mathsf{SK})$ such that $\mathsf{keygen}(H_0(\mathsf{SK}^*)) = \mathsf{keygen}(\mathsf{sk}^*)$ and $\mathsf{NKG}(\mathsf{SK}) \neq \mathsf{PK}^*$;
4. Guessed-SK^*-on-PK^*: $\mathcal{A}$ makes a query $P(\mathsf{pk}^*)$ with response $\mathsf{NKG}(\mathsf{SK}^*)$, and $\mathcal{A}$ also makes a query $H_0(\mathsf{SK}^*)$ such that $\mathsf{keygen}(H_0(\mathsf{SK}^*)) \neq \mathsf{pk}^*$.

For case 1, due to pseudorandom random public keys, it occurs with negligible probability. For case 2, as NKG is ideal NIKE oracle, collision never happens except with negligible probability.

For case 3, as the simulator samples sk^*, which is independent of $\mathcal{A}$'s view, the probability of $\mathcal{A}$ outputs a proper SK ($H_0(\mathsf{SK}) = \mathsf{sk}^*$) is bounded by pseudo-random public keys. Analogously, in case 4 $\mathcal{A}$ cannot guess SK^* correctly except for negligible probability.

Rule 4. Note that if the adversary first makes a query $H_0(\mathsf{SK})$, then the equation holds for certain when it requests P^{-1}. Hence, the only chance that $\mathcal{A}$ violates this rule is case 3 in Rule 3 occurs, referring to Rule 4 holds as long as Rule 3 holds.

Rule 5. Firstly, we note that the responses of H_1 queries are independent of the ones of H_0, P, P^{-1} queries, and the only consistency $\mathcal{S}$ has to preserve is the equation in this rule. Immediately observe that, if and only if, the inputs of the H_1 are in a good form (say, 1) $\mathsf{PK}_1 \leq \mathsf{PK}_2$; 2) shk is a valid shared key of pk_1 and pk_2), the response should be consistent to NSK-oracle (if the inputs are not in a good form, then the response is independent of NSK-oracle with high probability).

Easy note that, except for the last case (associated with underline), $\mathcal{S}$ responds to H_1 queries properly: $\mathcal{S}$ calls NSK-oracle when the input is within good form and otherwise returns a string $\mathsf{w} \in \mathcal{W}$. While, for the last one, the simulator just responds with w without checking whether the inputs are good or not. Hence, we hope that, for the last case, either the inputs are not in a good form or $\mathsf{NSK}(\mathsf{SK}_1, \mathsf{PK}_2)$ is independent of $\mathcal{A}$'s view. In fact, there are three bad cases that might break it:

1. A known secret key and another random public key: $\mathcal{A}$ chooses sk_1 and makes a query $P^{-1}(\mathsf{keygen}(\mathsf{sk}_1))$ with response $\mathsf{PK}_1 = \mathsf{NKG}(\mathsf{SK}_1)$, while $\mathsf{PK}_2 \notin T_P \cup T_{P^{-1}} \cup T_{H_0}$;
2. Two known secret keys in $T_{P^{-1}}$: Adversary chooses $\mathsf{sk}_1, \mathsf{sk}_2$ and makes queries $P^{-1}(\mathsf{keygen}(\mathsf{sk}_1)), \mathsf{P}^{-1}(\mathsf{keygen}(\mathsf{sk}_2))$ with responses $\mathsf{NKG}(\mathsf{SK}_1), \mathsf{NKG}(\mathsf{SK}_2)$;
3. Two known public keys without secret key keys: $\mathcal{A}$ chooses $\mathsf{SK}_1, \mathsf{SK}_2$ and makes queries $P(\mathsf{NKG}(\mathsf{SK}_1)), P(\mathsf{NKG}(\mathsf{SK}_2))$ with responses $\mathsf{pk}_1 = \mathsf{keygen}(\mathsf{sk}_1)$, $\mathsf{pk}_2 = \mathsf{keygen}(\mathsf{sk}_2)$.

For case 1, we note that PK_2 never appears in tables $T_P, T_{P^{-1}}$ and T_{H_0}, which means $P(\mathsf{PK}_2)$ is independent of $\mathcal{A}$'s view. Hence, the probability that shk is a valid shared key is bounded by the entropic shared keys (illustrated in Attack 6), which is negligible.

For case 2, we observe that $\mathcal{A}$ chooses the two secret keys itself; hence shk would be a valid shared key if $\mathcal{A}$ wants. However, SK_1 and SK_2 are independent of $\mathcal{A}$'s view, which refers to that $\mathsf{NSK}(\mathsf{SK}_1, \mathsf{PK}_2)$ is also independent of $\mathcal{A}$'s view.

For case 3, $\mathcal{A}$ knows pk_1 and pk_2 while the corresponding secret keys sk_1 and sk_2 are independent of $\mathcal{A}$'s view. Meanwhile, the adversary might implement $(\mathsf{NKG}, \mathsf{NSK})$ into oracles and use those oracles as an additional helper to predict the shared key (illustrated in Attack 7). Fortunately, $(\mathsf{NKG}, \mathsf{NSK})$ is an ideal NIKE; thus the only thing the adversary can do is *equality test*, which means the oracles adversary implements is the best helper it can count on. Hence, the probability that shk is a valid shared key is bounded by the semi-active unpredictable shared keys.

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) $|\mathcal{X}| \geq 2^{\omega(\log \lambda)}, |\mathcal{Y}| \geq 2^{\omega(\log \lambda)}, |\mathcal{R}| \geq 2^{\omega(\log \lambda)}$ *and* $|\mathcal{C}| \geq 2^{\omega(\log \lambda)}$*;* 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 $\mathcal{M} \cup \perp$. 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, then we say such a scheme is an ideal DPKE.

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: 1) convert this ideal NIKE into a PKE scheme; 2) apply the Fujisaki-Okamoto transformation [15], 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 $\mathcal{Z}$, and we denote $\mathsf{sk}, \mathsf{pk}, \mathsf{shk}$ to be the secret key, public key and shared key of Π_{NIKE}, respectively. For an easy exposition, we always denote the inputs of component primitives(for instance, in the standard model NIKE in Section 3 or the ideal NIKE in this section) as the lower-case and inputs of the target primitives as the upper-case. 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 $\mathsf{C} \stackrel{?}{=} \mathsf{DPKE.ENC}(\mathsf{PK}||\mathsf{M})$. If yes, then outputs A_3, else aborts.

Correctness easily follows, but this scheme is not indifferentiable. Next we present a differetiator to break it.

Differentiator for FO transform. Due to definition, $\mathcal{D}$ has three honest interfaces ($\mathsf{DPKE.KEYGEN}, \mathsf{DPKE.ENC}, \mathsf{DPKE.DEC}$)(below, we will denote ($\mathsf{DKG}, \mathsf{DE}$, DD) for short) and six adversarial interfaces ($H_0, H_1, H_2, H_3, \mathsf{NKG}, \mathsf{NSK}$), and we build $\mathcal{D}$ as in Figure 3:

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

Fig. 3: Differentiator 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, due to Π_{NIKE}'s correctness, 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 means that $\mathcal{D}$ always outputs 1. However, in the ideal game, the simulator knows nothing of queries to the honest interfaces, and the only information it has is: $(\mathsf{SK}, \mathsf{A}, \mathsf{B}_1, \mathsf{B}_2)$ (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 $\Pr[\mathcal{S}$ outputs a valid $\mathsf{B}_3] \leq \frac{\mathsf{poly}(\lambda)}{|\mathcal{M}|}$.

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 $\mathsf{C} \stackrel{?}{=} \mathsf{DPKE.ENC}(\mathsf{A}_3)$. If yes, then outputs A_4[9], else aborts.

In our new setting, we immediately observe that the ciphertext only consists of *two* elements, which means $\mathcal{S}$ can always hand in the vaild ciphertext to the decryption oracle. As a result, our new scheme prevents the attack above. 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 kind of efficient differetiators.

Theorem 7. (Indifferentiable DPKE). Π_{DPKE} *is indifferentiable from an ideal DPKE if* $\Pi_{\mathsf{NIKE}} = (\mathsf{NKG}, \mathsf{NSK})$ *is an ideal NIKE. More precisely, there exists a simulator* $\mathcal{S}$ *such that for all* $(q_{H_0}, q_{H_1}, q_P, q_{P^{-1}}, q_{\mathsf{NKG}}, q_{\mathsf{NSK}})$*-query PPT differentiator* $\mathcal{D}$ *with* $q_{H_0} + q_{H_1} + q_P + q_{P^{-1}} + q_{\mathsf{NKG}} + q_{\mathsf{NSK}} \leq q$*, we have*

$$\mathrm{Adv}^{\mathsf{indif}}_{\Pi_{\mathsf{DPKE}}, \mathcal{S}, \mathcal{D}} \leq \frac{11q^2}{|\mathcal{X}|} + \frac{20q^2}{|\mathcal{Y}|}.$$

The simulator makes at most q^2 *queries to its oracles.*

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 those attacks and complete the proof.

First attempt to build an indifferentiable PKE. Immediately to observe that we cannot build our scheme in the trivial way, say, treating the random nonce as part of the message and discarding it in the decryption procedure, because this construction loss the information of the randomness and that would induce a differentiator which trivially tests the validity of the randomness. To prevent it, we again apply the hash technique ($H_0 := \{0,1\}^* \to \mathcal{R}$); we first hash the nonce and then use the hashed value as the randomness. Specifically,

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

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

However, the scheme would not achieve indifferentiability if the random oracle is not well-designed, and the following we give out a differentiator to break our first attempt in Figure 4:

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. 4: Differentiator for non-well-designed random oracle.

Easy to note that, in real game, $\Pr[\mathcal{D} = 1] = 1$. Meanwhile, in ideal game, we claim that, with noticeable probability, the decryption would abort. In fact, the decryption procedure outputs M if and only if there exists a nonce $\mathsf{R} \in \mathcal{R}$ such that $H_0(\mathsf{PK}, \mathsf{M}, \mathsf{R}) = \mathsf{r}$. Moreover, $\mathsf{R}, \mathsf{r} \in \mathcal{R}$, and H_0 is a random oracle, we have that $\Pr[\forall \mathsf{R} \in \mathcal{R}, H_0(\mathsf{PK}, \mathsf{M}, \mathsf{R}) \neq r] \approx 1/e$, referring to $\Pr[\mathcal{D} = 1] \leq 1 - 1/e \approx 0.6$

Our solution. To prevent this attack, 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:
$$\mathsf{C} = \mathsf{DE}(\mathsf{PK}, \mathsf{M}||H_0(\mathsf{PK}, \mathsf{M}, \mathsf{R})||\underbrace{0 \ldots 0}_{n_3});$$
- $\mathsf{PKE.DEC}(\mathsf{SK}, \mathsf{C})$: 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, and the rest is to show its indifferentiability.

Theorem 8. (Indifferentiable PKE). Π_{PKE} *is indifferentiable from an ideal PKE if* $\Pi_{\mathsf{DPKE}} = (\mathsf{DKG}, \mathsf{DE}, \mathsf{DD})$ *is an ideal DPKE. More precisely, there exists a simulator such that for all* $(q_{H_0}, q_{\mathsf{DKG}}, q_{\mathsf{DE}}, q_{\mathsf{DD}})$*-query PPT differentiator* $\mathcal{D}$ *with* $q_{H_0} + q_{\mathsf{DKG}} + q_{\mathsf{DE}} + q_{\mathsf{DD}} \leq q$*, we have*

$$\mathrm{Adv}^{\mathsf{indif}}_{\Pi_{\mathsf{PKE}}, \mathcal{S}, \mathcal{D}} \leq 3q(\frac{1}{e})^{2^{n_3}} + \frac{6q^2}{2^{n_3}} + \frac{3q^2}{|\mathcal{C}|} + \frac{q}{|\mathcal{Y}| \times |\mathcal{M}|}.$$

The simulator makes at most q^2 queries to its oracles.

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 (random oracle model and ideal cipher model) 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) $|\mathcal{X}| \geq 2^{\omega(\log \lambda)}, |\mathcal{Y}| \geq 2^{\omega(\log \lambda)}, |\mathcal{M}| \geq 2^{\omega(\log \lambda)}$ *and* $|\Sigma| \geq 2^{\omega(\log \lambda)}$*;* 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 $\Pi_{\mathsf{Sig}} = (\mathsf{Sig.KEYGEN}, \mathsf{Sig.SIGN}, \mathsf{Sig.VER})$, 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_{Sig} is sampled from $\mathcal{T}$ uniformly.

5.2 Construction

In this section, we build our indifferentiable signature scheme from random oracle model and a standard-model signature scheme.

Difficulty of building an indifferentiable Signature. To achieve indifferentiability, we note that the signature σ would be masked by an idealized primitive, say $\Sigma = P^{-1}(\sigma)$ as above. In the verification algorithm, the scheme inverts Σ and uses σ to proceed. Unfortunately, this P-technique is insufficient here. In fact, given Σ (it is either a valid signature or a random string), the simulator cannot call Ver (the ideal signature) for help, as it does not know the public key and message. So the simulator cannot respond to $P(\Sigma)$ properly. To get rid of this difficulty, we would apply an ideal cipher model (E, E^{-1}) instead, where we set $(\mathsf{PK}, \mathsf{M})$ as its secret key. How does it help? We note that we force the adversary to hand it the public key and message to the simulator when inverting Σ; as a result, the simulator can call Ver and simulate $E(\Sigma)$ properly. Concretely,

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 $\mathcal{Z} \subset \{0,1\}^{n_3}$;
- $H_0 := \{0,1\}^* \to \mathcal{X}; H_1 := \{0,1\}^* \to \mathcal{M}$;
- $P := \{0,1\}^{n_1} \to \{0,1\}^{n_1}$ 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,

- $\mathsf{Sig.KEYGEN}(\mathsf{SK})$: On inputs secret key SK, the algorithm outputs public key $\mathsf{PK} = P^{-1}(\mathsf{keygen}(H_0(\mathsf{SK})))$;
- $\mathsf{Sig.SIGN}(\mathsf{SK}, \mathsf{M})$: On inputs secret key SK and message M, the algorithm computes $\mathsf{PK} = \mathsf{Sig.KEYGEN}(\mathsf{SK})$ 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 easily follows, and the rest is to prove its indifferentiability. Before that, we specify several 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 there is ϵ_1 such that for any PPT adversary $\mathcal{A}$, the advantage
$$\mathrm{Adv}_{\mathcal{A}} := \Pr[\mathsf{ver}(\mathsf{pk}, \mathsf{m}^*, \sigma^*) = 1 : \sigma^* \leftarrow \mathcal{A}^{\mathsf{sign}(\mathsf{sk}, \mathsf{m}_1, \ldots, \mathsf{m}_\mathsf{q})}(\mathsf{pk}, \mathsf{m}^*)] \leq \epsilon_1 = \mathsf{negl}(\lambda),$$
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 is pseudorandom, if there is ϵ_2 such that for any PPT adversary $\mathcal{A}$, $r \leftarrow \mathcal{X}, R \leftarrow \mathcal{Y}$.
$$\mathrm{Adv}_{\mathcal{A}} := |\Pr[\mathcal{A}(\mathsf{keygen}(r))] - \Pr[\mathcal{A}(R)]| \leq \epsilon_2 = \mathsf{negl}(\lambda).$$
We say a signature scheme is Good if it satisfies the three properties above.

Theorem 10. (Indifferentiable Signatures). Π_{Sig} *is indifferentiable from an ideal digital signature if* $\Pi_{\mathsf{SM-Sig}}$ *is* Good. *More preciously, there exists a simulator* $\mathcal{S}$ *such that for all* $(q_{H_0}, q_{H_1}, q_P, q_{P^{-1}}, q_E, q_{E^{-1}})$*-query PPT differentiator* $\mathcal{D}$ *with* $q_{H_0} + q_{H_1} + q_P + q_{P^{-1}} + q_E + q_{E^{-1}} \leq q$, *we have*
$$\mathrm{Adv}^{\mathsf{indif}}_{\Pi_{Sig}, \mathcal{S}, \mathcal{D}} \leq 15q^2 \sqrt{2\epsilon_2 + \frac{1}{|\mathcal{X}|}} + \frac{6q^2}{|\mathcal{Y}|} + \frac{8q}{|\mathcal{Z}|} + \frac{2q^2}{|\mathcal{M}|} + 6q^2\epsilon_1 + 3q^2\sqrt{\epsilon_2}.$$
The simulator makes at most q^2 *queries to its oracles.*

Acknowledgments

We thank Prof. Martin Albrecht for navigating the revisions of our paper as a shepherd. We thank the anonymous reviewers of Eurocrypt'2020 for pointing out a proof flaw in our earlier version. Mark Zhandry is supported by the Defense Advanced Research Projects Agency through the ARL under Contract W911NF-15-C-0205 and NSF grant 1616442. The views expressed are those of the authors and do not reflect the official policy or position of the Department of Defense or the National Science Foundation.

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. In *Annual International Cryptology Conference*, pages 187–220. Springer, 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. R. Canetti, O. Goldreich, and S. Halevi. The Random Oracle Methodology, Revisited. *J. ACM*, 51(4):557–594, July 2004.
8. 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.
9. 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.
10. 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.
11. W. Diffie and M. Hellman. New directions in cryptography. *IEEE transactions on Information Theory*, 22(6):644–654, 1976.
12. 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.
13. 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.
14. 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.

15. E. Fujisaki and T. Okamoto. Secure Integration of Asymmetric and Symmetric Encryption Schemes. *Journal of Cryptology*, 26(1):80–101, Jan 2013.
16. S. D. Galbraith, C. Petit, B. Shani, and Y. B. Ti. On the security of supersingular isogeny cryptosystems. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 63–91. Springer, 2016.
17. 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.
18. D. Jao and L. De Feo. Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies. In *International Workshop on Post-Quantum Cryptography*, pages 19–34. Springer, 2011.
19. 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.
20. 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.
21. 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.
22. 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.
23. 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.
24. 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.
25. C. E. Shannon. Communication theory of secrecy systems. *Bell system technical journal*, 28(4):656–715, 1949.
26. 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.
27. M. Zhandry and C. Zhang. Indifferentiability for public key cryptosystems. Cryptology ePrint Archive, Report 2019/370, 2019. `https://eprint.iacr.org/2019/370`.
28. 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.