Non-Interactive Key Exchange*

Eduarda S.V. Freire[1,**], Dennis Hofheinz[2,***], Eike Kiltz[3,†]
, and Kenneth G. Paterson[1,‡]

[1] Royal Holloway
[2] Karlsruhe Institute of Technology
[3] Ruhr-Universität Bochum

Abstract Non-interactive key exchange (NIKE) is a fundamental but much-overlooked cryptographic primitive. It appears as a major contribution in the ground-breaking paper of Diffie and Hellman, but NIKE has remained largely unstudied since then. In this paper, we provide different security models for this primitive and explore the relationships between them. We then give constructions for secure NIKE in the Random Oracle Model based on the hardness of factoring and in the standard model based on the hardness of a variant of the decisional Bilinear Diffie Hellman Problem for asymmetric pairings. We also study the relationship between NIKE and public key encryption (PKE), showing that a secure NIKE scheme can be generically converted into an IND-CCA secure PKE scheme. Our conversion also illustrates the fundamental nature of NIKE in public key cryptography.

Keywords: non-interactive key exchange, public-key cryptography, pairings.

1 Introduction

Non-interactive key exchange (NIKE) is a cryptographic primitive which enables two parties, who know each others' public keys, to agree on a symmetric shared key without requiring any interaction. The canonical example of a NIKE scheme can be found in the seminal paper by Diffie and Hellman [1]: let G be a group of prime order p with generator g, and assume Alice has public key $g^x \in G$ and private key $x \in \mathbb{Z}_p$, while Bob has public key $g^y \in G$ and private key $y \in \mathbb{Z}_p$. Then Alice and Bob can both compute the value $g^{xy} \in G$ without exchanging any messages. More properly, Alice and Bob should hash this key together with their identities in order to derive a symmetric key $H(\texttt{Alice}, \texttt{Bob}, g^{xy})$.

This example encapsulates in a nutshell all the basic features required of a NIKE scheme: users should agree on some common parameters (p, G and g here), then create their key pairs. Once these are computed and the public keys distributed, any pair of users can set up a shared key without further exchange of messages. The security properties desired of NIKE are, informally at least, clear: compromise of one user's private key should not affect the security of shared keys between pairs of uncorrupted users; compromise of one shared key should not undermine the security of other shared keys. Naturally, since the primitive is non-interactive, one cannot hope to obtain any kind of forward security properties. In practice, the public keys will be certified, and consideration needs to be given to modelling the key registration process.

NIKE has real-world applications. In wireless and sensor networks, conserving battery power is a prime concern, and so the energy cost of communication must be minimised. Thus using key establishment methods that minimise the number of bits that need to be transmitted is of fundamental importance. In particular, when faced with a jamming adversary, reducing the total number of rounds of interaction needed to establish a key is particularly helpful. NIKE is an excellent option in solving this problem, since a key can be established with minimal communication and interaction: assuming the public keys are pre-distributed, all that is needed is an exchange of identifiers for those keys, and often this exchange must take place anyway, in order to establish communications. A recent paper

---

* This is the full version of a paper with the same title to be presented at PKC 2013.

** Eduarda S.V. Freire was supported by CAPES Foundation/Brazil on grant 0560/09-0 and Royal Holloway, University of London.

*** Dennis Hofheinz was supported by a DFG grant (GZ HO 4534/2-1).

† Eike Kiltz was funded by a Sofja Kovalevskaja Award of the Alexander von Humboldt Foundation and the German Federal Ministry for Education and Research.

‡ Kenneth G. Paterson was supported by EPSRC Leadership Fellowship EP/H005455/1.

[2] gives a detailed evaluation of the energy costs of interactive and non-interactive key exchange protocols in the ID-based and PKI settings for wireless communications with a jamming adversary, demonstrating that significant energy savings can be made by adopting a non-interactive approach to key establishment. Its non-interactive nature makes NIKE an abstract building block that is *qualitatively* different from interactive key exchange: e.g., to achieve deniable authentication, [3] explicitly requires a *non-interactive* key exchange. But NIKE can also be used as a basis for *interactive* key exhange [4]: one can use the shared key in a MAC to authenticate an exchange of ephemeral Diffie-Hellman values. Finally, NIKE can be used to build very simple non-interactive designated verifier signature schemes [5], again using the shared key in a MAC to authenticate messages. Thus NIKE appears in various guises throughout the literature.

Despite its appearing in the very first paper on public key cryptography, the NIKE primitive has so far received scant attention as a primitive in its own right. Cash, Kiltz and Shoup (CKS) [6] provided a basic security model for NIKE and analysed the Diffie-Hellman-based scheme above, as well as a twinned variant of it, in the Random Oracle Model (ROM). There is also some work in the ID-based setting [7,8,9,10], also all restricted to the ROM.

Our contributions: Our contention is that NIKE is long overdue for more serious attention and development. In this paper, we initiate the systematic study of NIKE in the public key setting, providing: models and their relationships; constructions for secure NIKE in the Random Oracle Model and in the standard model in the challenging setting where the adversary can introduce arbitrary public keys into the system; and a construction for IND-CCA secure public key encryption (PKE) from any secure NIKE. Let us expand on each of these contributions in turn.

Models: It would seem that definitions and security models for interactive key exchange (e.g., [11,12,13,14]) could provide a natural starting point for formalising NIKE. However, here we take the CKS definition [6] for NIKE as our starting point. One reason for using a case-tailored NIKE definition is simplicity: existing security models for interactive key exchange give considerable attention to properties which are irrelevant in the NIKE setting. (For instance, forward security, multiple sessions, and in particular the pairing of sessions play no role in a non-interactive setting.) Another reason for a case-tailored NIKE definition is that we can focus on adversarial key registration queries; these are usually only implicitly [14] (or not at all [11,13]) considered in the standard models for interactive key exchange[4]. However, in our setting, adversarial key registrations pose the main technical obstacle to achieve NIKE security, as we will explain below.

The CKS security model for NIKE uses an indistinguishability- and game-based approach to define security, with the adversary being required to distinguish real from random keys in responses to its test queries. The model does allow the adversary to register public keys of his choice in the system and then to make queries for the shared keys between these "corrupted" users and honest (non-adversarially controlled) users, so-called *corrupt reveal queries.* This translates in the real world to minimising the assumptions made about certification procedures followed by the Certification Authority (CA) in the PKI supporting the NIKE: it means that the CA is not assumed to check that a public key submitted for certification has not been submitted before, and does not check that the party submitting the public key knows the corresponding private key. The model for NIKE in [6] is similar to, and presumably inspired by, the early work of Shoup [12] on interactive key exchange, where capturing so-called PKI attacks, also known as rogue-key attacks, was intrinsic to the security modelling. This modelling approach is referred to elsewhere in the literature as the *plain* setting (see [16,17] and the references therein) or the *bare PKI* setting [3]. The CKS model is certainly more challenging than settings where proofs of knowledge or proofs of possession of private keys are assumed to be given during registration, or where the adversary must reveal its secret key directly (as with the knowledge of secret key assumption used in [18,19]). However, the CKS model has some shortcomings: the adversary is not allowed to directly query for the shared keys held between pairs of honest users, but instead only gets to see real or random values for these via test queries. Moreover the model does not allow an adversary to query for the private keys of honestly registered users.

Therefore, as a necessary precursor to the further development of NIKE, we start by exploring different models for NIKE and their relationships (Section 2). In summary, we introduce three new security models for NIKE and show that they are all polynomially equivalent to one another and to

[4] We mention that some security analyses (e.g., [15]) and Shoup's security model [12] do explicitly consider adversarial key registration queries.

the original CKS model from [6]. One of our models, the *m-CKS-heavy* model, augments the CKS model and effectively allows all conceivable queries, without allowing the adversary to win trivially. It is our preferred security model for NIKE. Another of our models, *CKS-light*, allows only two honest users, no corruption of honest users, and a single test query. Thus it is particularly simple and so easy to use when analyzing specific NIKE schemes; moreover our results showing equivalence between the models ensure that security in this model implies security in the preferred *m-CKS-heavy* model.

We stress that all these models allow the adversary to register public keys of his choice in the system, so are in the plain setting. However, for completeness, we also briefly consider the *HKR* or *honest key registrations* setting in which the adversary cannot register keys on its own. It is easy to see that the *HKR* setting provides strictly weaker security guarantees than our default security setting with dishonest key registrations. For instance, the already mentioned Diffie-Hellman NIKE scheme *without hashing* (such that shared keys are of the form g^{xy}) can be shown secure in the *HKR* setting under the Decisional Diffie-Hellman assumption, but is easily seen to be completely insecure in our default setting[5].

Constructions for NIKE: In Section 4, we give two concrete constructions for NIKE schemes meeting our *CKS-light* security definition, and hence secure in our preferred *m-CKS-heavy* model (*with* dishonest key registrations). Our two constructions are inspired by public key encryption (PKE) schemes which are secure against chosen-ciphertext attacks (IND-CCA secure). We note that dealing with *corrupt reveal* queries requires techniques to guard against active attacks, which in part explains the connection to IND-CCA security. Indeed, we will also show how to go in the reverse direction, converting any secure NIKE scheme into an IND-CCA secure PKE scheme, see below. We stress, however, that we cannot simply take any IND-CCA secure PKE scheme and directly interpret it as a NIKE scheme.[6] Rather, our constructions for NIKE exploit specific properties of the underlying PKE schemes. In fact, our belief is that a generic construction for secure NIKE from PKE is unlikely to be forthcoming.

The first scheme acts as a warm-up. It is provably secure under the factoring assumption in the Random Oracle Model (ROM) and uses ideas from [20] to analyse the basic Diffie-Hellman scheme, where keys are of the form $H(\texttt{Alice}, \texttt{Bob}, g^{xy})$, in the group of signed quadratic residues. We note that closely related schemes were analysed in [6], but in different groups and under different assumptions. Specifically, a twinned version of the scheme was proved secure under the CDH assumption, while it is stated that the basic Diffie-Hellman scheme is secure under the Strong DH assumption. We remark that the latter claim of [6] is problematic. Concretely, the Strong DH assumption is not (directly) sufficient to show that the basic Diffie-Hellman scheme is secure. Namely, the corresponding security reduction requires *two* DDH oracles – one for each of the two users sharing the key on which the adversary wants to be challenged – while the Strong DH assumption supplies only one. Certainly this problem could be solved instead by appealing to a suitable gap-DH assumption. We show how to overcome this problem in the group of signed quadratic residues without the need to rely on a gap assumption. We then proceed to sketch how to transport this scheme to the standard model, under the additional assumption that the adversary only registers valid public keys. Because of the extra assumption, this scheme does not strictly speaking meet our security definitions, and would require validity to be enforced by some means in an interactive registration protocol (for example, via a proof of correctness of the public key). This limitation of our standard model, factoring-based solution reflects the technical challenge involved in achieving our "bare PKI" security notions.

Our second NIKE scheme is provably secure in the standard model and combines a specific weak Programmable Hash Function [21] whose output lies in a pairing group and a Chameleon hash function. This enables the simulation in our security proof for the scheme to handle the tricky queries for shared keys involving an honestly generated public key and an adversarially chosen public

[5] Concretely, since shared keys do not depend on party identities in the unhashed DH-NIKE, an adversary $\mathcal{A}$ can (a) register the key g^x of an honest party Alice as its own key, and (b) ask for the shared key between $\mathcal{A}$ and another honest party Bob with key g^y. This immediately yields the shared key g^{xy} between Alice and Bob. Because of the homomorphic properties of the DH-NIKE, a simple modification of this attack also works if $\mathcal{A}$ is not allowed to register keys of existing users.

[6] One reason is that it is not clear what should correspond to the NIKE public key: a PKE public key, a PKE ciphertext, or a combination of both? Besides, the corresponding security experiments for NIKE and PKE schemes are rather different: there usually is one challenge ciphertext in a PKE security experiment, while there are at least two challenge users in a NIKE security experiment.

key. Similar ideas were used in the context of HIBE in [22]. We also make use of the pairing to provide a means of checking that public keys coming from the adversary are in some sense well-formed. We work with asymmetric pairings for efficiency at high security levels (and because it does not add any real complexity to the description of our scheme). The scheme's security relies on a natural variant of the Decisional Bilinear Diffie-Hellman (DBDH) assumption for the asymmetric setting.

From NIKE to PKE: In Section 5, we explore the connections between NIKE and public key encryption (PKE). That such connections exist should not be too much of a surprise: it is folklore that the ElGamal encryption scheme [23] can be seen as arising from the Diffie-Hellman NIKE scheme by making the sender's key pair (g^x, x) ephemeral and using the receiver's public key g^y to create the basis for a shared key g^{xy}. In fact, a simple transformation shows that every NIKE that is secure in the simpler *HKR* setting can be turned into a public key encryption scheme that is secure against chosen-plaintext attacks (IND-CPA secure). Similar connections were explored in the ID-based setting in [10].

In our default setting with dishonest key registrations, we provide a simple, generic construction for PKE from NIKE that is also in the spirit of the original Diffie-Hellman–to–ElGamal conversion. The construction takes a NIKE scheme that is secure in our *CKS-light* model (with dishonest key registrations) and a strongly one-time secure signature scheme as inputs, and produces from these components a Key Encapsulation Mechanism (KEM) that we prove to be IND-CCA secure. A secure PKE from such a KEM can be obtained using standard results. At a high level, the key pair for the KEM is a randomly generated key pair (pk, sk) from the NIKE scheme, ciphertexts are also randomly generated public keys pk' from the NIKE scheme (together with a one-time signature that binds the public key to an identity), while the encapsulated key is the shared key computed from sk' and pk; the receiver computes the same key from sk and pk', assuming the one-time signature verifies. In order to prove the KEM to be IND-CCA secure, we exploit the presence of corrupt reveal queries in the NIKE security model in an essential way to handle certain decapsulation queries. The resulting KEM is almost as efficient as the underlying NIKE scheme. In the *HKR* setting, the same transformation (only without one-time signatures) shows that *CKS-light* security of the NIKE implies IND-CPA security of the resulting PKE scheme.

The fact that secure NIKE implies IND-CCA-secure PKE, one of the most important primitives in cryptography, illustrates the fundamental role and utility of NIKE. We believe that this connection should spur further research on the topic.

2 Non-interactive Key Exchange and Security Models

2.1 Non-interactive Key Exchange

Following [6], we formally define a Non-Interactive Key Exchange (NIKE) scheme in the public key setting to be a collection of three algorithms: `CommonSetup`, `NIKE.KeyGen` and `SharedKey` together with an identity space $\mathcal{IDS}$ and a shared key space $\mathcal{SHK}$. Note that identities in the scheme and security model are merely used to track which public keys are associated with which users – we are *not* in the identity-based setting.

- `CommonSetup`: On input 1^k, outputs *params*, a set of system parameters.
- `NIKE.KeyGen`: On input *params* and an identity $\mathsf{ID} \in \mathcal{IDS}$, outputs a public key/secret key pair (pk, sk). This algorithm is probabilistic and can be executed by any user. We assume, without loss of generality, that *params* is included in *pk*.
- `SharedKey`: On input an identity $\mathsf{ID}_1 \in \mathcal{IDS}$ and a public key pk_1 along with another identity $\mathsf{ID}_2 \in \mathcal{IDS}$ and a secret key sk_2, outputs either a shared key in $\mathcal{SHK}$ for the two identities, or a failure symbol $\perp$. This algorithm is assumed to always output $\perp$ if $\mathsf{ID}_1 = \mathsf{ID}_2$.

For correctness, we require that, for any pair of identities $\mathsf{ID}_1, \mathsf{ID}_2$, and corresponding key pairs (pk_1, sk_1) and (pk_2, sk_2), algorithm `SharedKey` satisfies the constraint:

$$\texttt{SharedKey}(\mathsf{ID}_1, pk_1, \mathsf{ID}_2, sk_2) = \texttt{SharedKey}(\mathsf{ID}_2, pk_2, \mathsf{ID}_1, sk_1).$$

2.2 Definitions of Security for Non-interactive Key Exchange

Cash, Kiltz and Shoup [6] proposed a security model for NIKE schemes in the public key setting, denoted here by the *CKS* model. This model abstracts away all considerations concerning certification and PKI in a particularly nice way. It allows an adversary to obtain honestly generated public keys, but also to then associate such public keys with other identities, and to register dishonestly generated public keys (for which the adversary need not know the corresponding private keys). This dishonest key registration (DKR) setting (abstractly) models a PKI where minimal assumptions are made about the actions of the Certificate Authority (CA): the CA is not assumed to check that a public key has not been previously registered to another user, and does not demand a proof of knowledge or possession of the private key when issuing a certificate on a public key. This conservative approach to modelling is fully appropriate given the great diversity in how CAs operate in the real world. The model can be seen as a natural adaptation of the approach of Shoup [12] for modelling interactive key exchange to the NIKE setting and is analogous to the plain setting studied in [16,17].

However, there are some obvious omissions from the model, including the ability of an adversary to "corrupt" honestly generated public keys to learn the corresponding private keys, and the ability of a user to directly learn the key shared between two honest parties in the system (which could be possible, for example, because of cryptanalysis of a scheme making use of the shared key). Equivalent queries in the ID-based setting were permitted in the model introduced in [10].

For this reason, we augment the original *CKS* model with the "missing" queries, introducing the *m-CKS-heavy* model. We regard this as providing the "correct" model for NIKE. We also introduce two further models, the *CKS-heavy* and *CKS-light* models. These differ from *m-CKS-heavy* and the original *CKS* model only in the numbers and types of query that the adversary is allowed to make. Next we present in detail the *m-CKS-heavy* model. Then in Table 1 we summarize the differences between these security models in the DKR setting.

The m-CKS-heavy model: Our model is stated in terms of a game between an adversary $\mathcal{A}$ and a challenger $\mathcal{C}$. In this game, $\mathcal{C}$ takes as input the security parameter 1^k, runs algorithm `CommonSetup` of the NIKE scheme and gives $\mathcal{A}$ *params*. The challenger takes a random bit b and answers oracle queries for $\mathcal{A}$ until $\mathcal{A}$ outputs a bit $\hat{b}$. The challenger answers the following types of queries for $\mathcal{A}$:

- *Register honest user ID*: $\mathcal{A}$ supplies an identity $\mathsf{ID} \in \mathcal{IDS}$. On input *params* and ID, the challenger runs `NIKE.KeyGen` to generate a public key/secret key pair (pk, sk) and records the tuple $(honest, \mathsf{ID}, pk, sk)$. The challenger returns pk to $\mathcal{A}$.
- *Register corrupt user ID*: In this type of query, $\mathcal{A}$ supplies both an identity $\mathsf{ID} \in \mathcal{IDS}$ and a public key pk. The challenger records the tuple $(corrupt, \mathsf{ID}, pk, \perp)$. We stress that $\mathcal{A}$ may make multiple "Register corrupt user ID" queries for the same ID during the experiment. In that case, only the most recent $(corrupt, \mathsf{ID}, pk, \perp)$ entry is kept.
- *Extract queries*: Here $\mathcal{A}$ supplies an identity ID that was registered as an honest user. The challenger looks for a tuple $(honest, \mathsf{ID}, pk, sk)$ containing ID and returns sk to $\mathcal{A}$.
- *Reveal queries*: Here $\mathcal{A}$ supplies a pair of registered identities $\mathsf{ID}_1, \mathsf{ID}_2$, subject only to the restriction that at least one of the two identities was registered as *honest*. The challenger runs `SharedKey` using the secret key of one of the *honest* identities and the public key of the other identity and returns the result to $\mathcal{A}$. Note that here the adversary is allowed to make reveal queries between two users that were originally registered as honest users. We denote by *honest reveal* the queries involving two honest users and by *corrupt reveal* the queries involving an honest user and a corrupt user.
- *Test queries*: Here $\mathcal{A}$ supplies two distinct identities $\mathsf{ID}_1, \mathsf{ID}_2$ that were both registered as honest. The challenger returns $\perp$ if $\mathsf{ID}_1 = \mathsf{ID}_2$. Otherwise, it uses the bit b to answer the queries. If $b = 0$, the challenger runs `SharedKey` using the public key for ID_1 and the secret key for ID_2 and returns the result to $\mathcal{A}$. If $b = 1$, the challenger generates a random key, records it for later, and returns that key to the adversary. In this case, to keep things consistent, the challenger returns the same random key for the pair $\mathsf{ID}_1, \mathsf{ID}_2$ every time $\mathcal{A}$ queries for their paired key, in either order.

$\mathcal{A}$'s queries may be made adaptively and are arbitrary in number. To prevent trivial wins for the adversary, no query to the *reveal* oracle is allowed on any pair of identities selected for *test* queries (in either order), and no *extract* query is allowed on any of the identities involved in *test* queries.

Model	Register Honest	Register Corrupt	Extract	Honest Reveal	Corrupt Reveal	Test
CKS-light	2	✓	✗	✗	✓	1
CKS	✓	✓	✗	✗	✓	✓
CKS-heavy	✓	✓	✓	✓	✓	1
m-CKS-heavy	✓	✓	✓	✓	✓	✓

Table 1. Types of queries for different security models in the dishonest key registration (DKR) PKI model (aka plain/bare model). Notation: ✓means that an adversary is allowed to make an arbitrary number of queries; ✗ means that no queries can be made; numbers represent the number of queries allowed to an adversary.

Also, we demand that no identity registered as corrupt can later be the subject of a *register honest user ID* query, and vice versa.

When the adversary finally outputs $\hat{b}$, it wins the game if $\hat{b} = b$. For an adversary $\mathcal{A}$, we define its advantage in this security game as:

$$\mathrm{Adv}_{\mathcal{A}}^{\text{m-CKS-heavy}}(k, q_H, q_C, q_E, q_{HR}, q_{CR}, q_T) = |\Pr[\hat{b} = b] - 1/2|$$

where q_H, q_C, q_E, q_{HR}, q_{CR} and q_T are the numbers of *register honest user ID* queries, *register corrupt user ID* queries, *extract* queries, *honest reveal* queries, *corrupt reveal* queries and *test* queries made by $\mathcal{A}$, respectively. We say that a NIKE scheme is $(t, \epsilon, q_H, q_C, q_E, q_{HR}, q_{CR}, q_T)$-secure in the *m-CKS-heavy* model if there is no adversary with advantage at least ϵ that runs in time t and makes at most q_H *register honest user ID* queries, etc. Informally, we say that a NIKE scheme is *m-CKS-heavy secure* if there is no efficient adversary having non-negligible advantage in k, where efficient means that the running time and numbers of queries made by the adversary are bounded by polynomials in k.

Comparing the models: Table 1 outlines the properties of our other security models in the DKR setting, in terms of restrictions on the queries that can be made by the adversary. It is apparent that the *m-CKS-heavy* model is the strongest model. It differs from the *CKS-heavy* model only in allowing multiple *test* queries. The *m-CKS-heavy* model represents a strengthening of the original *CKS* model by allowing *extract* and *honest reveal* queries, whereas the *CKS* model only allows the adversary to gain information about honestly generated shared keys via *test* queries. The *CKS-light* model is simplest of all, involving only two honestly registered identities, removing the *extract* and *honest reveal* queries, and allowing only a single *test* query. We prove in Appendix B that it is polynomially equivalent to the *m-CKS-heavy* model. In fact, we prove there the following theorem:

Theorem 1. *The* m-CKS-heavy, CKS-heavy*,* CKS *and* CKS-light *security models are all polynomially equivalent. More specifically, for any scheme* `NIKE`*, we have the following results (where advantages for the* CKS-heavy*,* CKS *and* CKS-light *security models are defined in the obvious way):*

CKS-heavy ⇒ m-CKS-heavy: For any adversary $\mathcal{A}$ against `NIKE` *in the* m-CKS-heavy *model, there is an adversary $\mathcal{B}$ that breaks* `NIKE` *in the* CKS-heavy *model with*

$$\mathrm{Adv}_{\mathcal{B}}^{\text{CKS-heavy}}(k, q_H, q_C, q_E, q'_{HR}, q_{CR}) = \mathrm{Adv}_{\mathcal{A}}^{\text{m-CKS-heavy}}(k, q_H, q_C, q_E, q_{HR}, q_{CR}, q_T)/q_T,$$

where $q'_{HR} \leq q_{HR} + q_T$. The converse, CKS-heavy ⇐ m-CKS-heavy is trivial.

CKS-light ⇒ CKS: For any adversary $\mathcal{A}$ against `NIKE` *in the* CKS *model, there is an adversary $\mathcal{B}$ that breaks* `NIKE` *in the* CKS-light *model with*

$$\mathrm{Adv}_{\mathcal{B}}^{\text{CKS-light}}(k, q'_C, q'_{CR}) \geq 2 \cdot \mathrm{Adv}_{\mathcal{A}}^{\text{CKS}}(k, q_H, q_C, q_{CR}, q_T)/{q_H}^2 q_T,$$

where $q'_C \leq q_C + q_H$ and $q'_{CR} \leq q_{CR}$. The converse, CKS-light ⇐ CKS is trivial.

CKS-light ⇒ CKS-heavy: For any adversary $\mathcal{A}$ against `NIKE` *in the* CKS-heavy *model, there is an adversary $\mathcal{B}$ that breaks* `NIKE` *in the* CKS-light *model with*

$$\mathrm{Adv}_{\mathcal{B}}^{\text{CKS-light}}(k, q'_C, q'_{CR}) \geq 2 \cdot \mathrm{Adv}_{\mathcal{A}}^{\text{CKS-heavy}}(k, q_H, q_C, q_E, q_{HR}, q_{CR})/{q_H}^2,$$

where $q'_C \leq q_C + q_H$ and $q'_{CR} \leq q_{CR}$. The converse, CKS-light ⇐ CKS-heavy is trivial.

Model	Register Honest	Register Corrupt	Extract	Honest Reveal	Corrupt Reveal	Test
HKR CKS-light	2	✗	✗	✗	✗	1
HKR CKS	✓	✗	✗	✗	✗	✓
HKR CKS-heavy	✓	✗	✓	✓	✗	1
HKR m-CKS-heavy	✓	✗	✓	✓	✗	✓

Table 2. Types of queries for different security models in the honest key registration (HKR) PKI model.

Thus, while the *m-CKS-heavy* model is our preferred model, it suffices to analyse schemes in the *CKS-light* model if one is not overly concerned about concrete security. However, we note that various factors are involved in the reductions. In particular a factor of $q_T {q_H}^2$ is lost in going from the *m-CKS-heavy* to the *CKS-light* model. This reflects the proof techniques used in establishing the bounds, specifically the use of hybrid arguments. It is an interesting open problem to either prove tighter relations between the models, or to prove that such results are not possible.

Security models in the honest key registration (HKR) setting: For completeness we also provide NIKE security models in the honest key registration setting where dishonest key registration queries are disallowed. An overview of the models is given in Table 2. We remark that Theorem 1 carries over to the HKR setting simply by setting q_C and q_{CR} to zero in the theorem statement and proofs. So all the security models from Table 2 are equivalent to one another. As pointed out in the introduction, constructing NIKE schemes in the HKR setting is much easier than in the more realistic DKR setting.

3 Intractability Assumptions

3.1 The Group of Signed Quadratic Residues, the BBS generator, and the Strong Diffie-Hellman Assumption

The factoring assumption: Let $n(k)$ be a function and δ a constant with $0 \leq \delta < 1/2$. Let $\mathtt{RSAgen}$ be an algorithm with input 1^k that generates elements (N, P, Q) such that $N = PQ$ is an n-bit Blum integer and all prime factors of $\phi(N)/4$ are pairwise distinct and have at least δn bits. These conditions ensure that $(\mathbb{J}_N, \cdot)$ is cyclic and that the square g of a random element in $\mathbb{Z}_N^*$, generates $\mathbb{QR}_N$ with high probability. That is, $\langle g \rangle = \mathbb{QR}_N$. For such N, we recall the definition of the *group of signed quadratic residues* $\mathbb{QR}_N^+$ from [20] (see also [24,25]) which is defined as the set $\{|x| : x \in \mathbb{QR}_N\}$, where $|x|$ is the absolute value when representing elements of $\mathbb{Z}_N$ as the set $\{-(N-1)/2, \ldots, (N-1)/2\}$. $(\mathbb{QR}_N^+, \cdot)$ is a cyclic group of order $\phi(N)/4$ whose elements are efficiently recognisable given only N as input.

For any algorithm $\mathcal{A}$, we write

$$\mathrm{Adv}^{\mathrm{fac}}_{\mathcal{A},\mathtt{RSAgen}}(k) = \Pr[\{P, Q\} \xleftarrow{\$} \mathcal{A}(N) : (N, P, Q) \xleftarrow{\$} \mathtt{RSAgen}(\mathtt{1^k})].$$

The factoring assumption for $\mathtt{RSAgen}$ is that $\mathrm{Adv}^{\mathrm{fac}}_{\mathcal{A},\mathtt{RSAgen}}(k)$ is negligible for all PPT algorithms $\mathcal{A}$.

The BBS generator: Let $\mathrm{BBS}_N : \mathbb{QR}_N^+ \to \{0,1\}^k$ be the Blum-Blum-Shub pseudorandom number generator. (That is, $\mathrm{BBS}_N(X) = (\mathrm{lsb}_N(X), \mathrm{lsb}_N(X^2), \ldots, \mathrm{lsb}_N(X^{2^{k-1}}))$, where $\mathrm{lsb}_N(X)$ denotes the least significant bit of $X \in \mathbb{QR}_N^+$.) Recall that the factoring assumption implies the computational indistinguishability of the distributions

$$(N, X^{2^k}, \mathrm{BBS}_N(X)) \text{ and } (N, X^{2^k}, R),$$

where $N \xleftarrow{\$} \mathtt{RSAgen}(\mathtt{1^k})$, and $X \xleftarrow{\$} \mathbb{QR}_N^+$ and $R \xleftarrow{\$} \{0,1\}^k$ are chosen uniformly. (See also [26, Theorem 2] for a summary why this holds.) Concretely, under the factoring assumption, the advantage

$$\mathrm{Adv}^{\mathrm{BBS}}_{\mathcal{B},\mathtt{RSAgen}}(k) := \left| \Pr[\mathcal{B}(N, X^{2^k}, \mathrm{BBS}_N(X)) = 1] - \Pr[\mathcal{B}(N, X^{2^k}, R) = 1] \right|$$

is negligible for any PPT adversary $\mathcal{B}$.

The Strong DH assumption: In [20] it is shown that if the factoring assumption holds, then the *Strong DH assumption* holds relative to $\texttt{RSAgen}$. This assumption is that there is no PPT algorithm having non-negligible advantage in solving the CDH problem on input (N, g, X, Y) when given an oracle for $\mathrm{DDH}_{g,X}(\cdot,\cdot)$. Here g is a randomly selected generator of $\mathbb{QR}_N^+$, X and Y are selected uniformly from $\mathbb{QR}_N^+$, the solution to the CDH problem is defined as $g^{(\mathrm{dlog}_g X)(\mathrm{dlog}_g Y)}$, and the DDH oracle $\mathrm{DDH}_{g,X}(\hat{Y}, \hat{Z})$ returns 1 if $\hat{Y}^{\mathrm{dlog}_g X} = \hat{Z}$ and 0 otherwise.

We will require a variant of the Strong DH assumption, which we name the *Double Strong DH (DSDH)* assumption. This can be stated as follows. Let $(N, P, Q) \leftarrow \texttt{RSAgen}(\texttt{1}^\texttt{k})$ and let g be a randomly selected generator of $\mathbb{QR}_N^+$, and X, Y be selected uniformly from $\mathbb{QR}_N^+$. Then the Double Strong DH problem is to solve the CDH problem on input (N, g, X, Y), that is to compute $g^{(\mathrm{dlog}_g X)(\mathrm{dlog}_g Y)}$, when given oracles for $\mathrm{DDH}_{g,X}(\cdot,\cdot)$ and $\mathrm{DDH}_{g,Y}(\cdot,\cdot)$. The DSDH assumption relative to $\texttt{RSAgen}$ is that there is no PPT algorithm having non-negligible advantage in solving this problem.

Theorem 2. *If the factoring assumption holds relative to* $\texttt{RSAgen}$*, then the DSDH assumption also holds relative to* $\texttt{RSAgen}$*. In particular, for every algorithm $\mathcal{A}$ solving the Double Strong DH problem, there exists a factoring algorithm $\mathcal{B}$ (with roughly the same running time as $\mathcal{A}$) such that*

$$\mathrm{Adv}^{\mathrm{dsdh}}_{\mathcal{A},\texttt{RSAgen}}(k) \leq \mathrm{Adv}^{\mathrm{fac}}_{\mathcal{B},\texttt{RSAgen}}(k) + O(2^{-\delta n(k)}).$$

Proof. The original proof of [20, Theorem 2] shows how to handle a single DDH oracle $\mathrm{DDH}_{g,X}(\cdot,\cdot)$. By symmetry of the set-up used in the proof, the same procedure can also be used to (simultaneously) handle the oracle $\mathrm{DDH}_{g,Y}(\cdot,\cdot)$.

3.2 Parameter generation algorithms for Asymmetric Pairings

Our pairing based scheme will be parameterized by a *type 2 pairing parameter generator*, denoted by $\mathcal{G}2$. This is a polynomial time algorithm that on input a security parameter 1^k, returns the description of three multiplicative cyclic groups $\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{G}_T$ of the same prime order p, generators g_1, g_2 for $\mathbb{G}_1$, $\mathbb{G}_2$ respectively, and a bilinear non-degenerate and efficiently computable pairing $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$. We assume that $\mathcal{G}2$ also outputs the description of an efficiently computable isomorphism $\psi : \mathbb{G}_2 \to \mathbb{G}_1$ and that $g_1 = \psi(g_2)$. Throughout, we write $\mathcal{PG}2 = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2, p, e, \psi)$ for a set of groups and other parameters with the properties just described.

3.3 The Decisional Bilinear Diffie-Hellman Assumption for Type 2 Pairings (DBDH-2)

Let $\mathcal{PG}2 = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2, p, e, \psi)$ as above. We consider the following version of the Decisional Bilinear Diffie-Hellman problem for type 2 pairings, as introduced by Galindo in [27]: Given $(g_2, g_2^a, g_2^b, g_1^c, T) \in \mathbb{G}_2^3 \times \mathbb{G}_1 \times \mathbb{G}_T$ as input, the problem is to decide whether or not $T = e(g_1, g_2)^{abc}$, where $g_1 = \psi(g_2)$. More formally, we associate the following experiment to a type 2 pairing parameter generator $\mathcal{G}2$ and an adversary $\mathcal{B}$.

Experiment $\mathrm{Exp}^{\text{dbdh-2}}_{\mathcal{B},\mathcal{G}2}(k)$
$\quad \mathcal{PG}2 \xleftarrow{\$} \mathcal{G}2(1^k)$
$\quad a, b, c, z \xleftarrow{\$} \mathbb{Z}_p$
$\quad \beta \xleftarrow{\$} \{0, 1\}$
$\quad$ If $\beta = 1$ then $T \leftarrow e(g_1, g_2)^{abc}$ else $T \leftarrow e(g_1, g_2)^z$
$\quad \beta' \xleftarrow{\$} \mathcal{B}(1^k, \mathcal{PG}2, g_2^a, g_2^b, g_1^c, T)$
$\quad$ If $\beta = \beta'$ then return 0 else return 1

The advantage of $\mathcal{B}$ in the above experiment is defined as

$$\mathrm{Adv}^{\text{dbdh-2}}_{\mathcal{B},\mathcal{G}2}(k) = \left| \Pr[\mathrm{Exp}^{\text{dbdh-2}}_{\mathcal{B},\mathcal{G}2}(k) = 1] - \frac{1}{2} \right|.$$

We say that the DBDH-2 assumption relative to $\mathcal{G}2$ holds if $\mathrm{Adv}^{\text{dbdh-2}}_{\mathcal{B},\mathcal{G}2}$ is negligible in k for all PPT algorithms $\mathcal{B}$.

4 Constructions for Non-interactive Key Exchange

4.1 A Construction in the Random Oracle Model from Factoring

We specify how to build a NIKE scheme, $\mathrm{NIKE}_{\mathrm{fac}}$, that is secure in the *CKS-light* security model under the factoring assumption relative $\mathtt{RSAgen}$ in the ROM. Our scheme makes use of a hash function $H : \{0,1\}^* \to \{0,1\}^k$ which is modelled as a random oracle in the security proof. The component algorithms of the scheme $\mathrm{NIKE}_{\mathrm{fac}}$ are defined as follows:

$\mathtt{CommonSetup}(1^k)$	$\mathtt{NIKE.KeyGen}(params, \mathsf{ID})$
$(N,P,Q) \xleftarrow{\$} \mathtt{RSAgen(1^k)}$	$x \xleftarrow{\$} \mathbb{Z}_{\lfloor N/4 \rfloor}$;
$g \xleftarrow{\$} \mathbb{QR}_N^+$, where $\langle g \rangle = \mathbb{QR}_N^+$	$X \leftarrow g^x$
$params \leftarrow (H,N,g)$	$pk \leftarrow X$; $sk \leftarrow x$
Return $params$	Return (pk, sk)

$\mathtt{SharedKey}(\mathsf{ID}_1, pk_1, \mathsf{ID}_2, sk_2)$
If $(\mathsf{ID}_1 = \mathsf{ID}_2)$ or $pk_1 \notin \mathbb{QR}_N^+$ or $pk_2 \notin \mathbb{QR}_N^+$ return $\perp$

$$\text{else if } \begin{cases} \mathsf{ID}_1 < \mathsf{ID}_2 \text{ return } H(\mathsf{ID}_1, \mathsf{ID}_2, {pk_1}^{sk_2}) \\ \mathsf{ID}_2 < \mathsf{ID}_1 \text{ return } H(\mathsf{ID}_2, \mathsf{ID}_1, {pk_1}^{sk_2}) \end{cases}$$

Here we are assuming that the identities ID come from a space with a natural ordering $<$.

Theorem 3. *The scheme* $\mathrm{NIKE}_{\mathrm{fac}}$ *is secure in the ROM under the factoring assumption relative to* $\mathtt{RSAgen}$. *In particular, suppose $\mathcal{A}$ is an adversary against* $\mathrm{NIKE}_{\mathrm{fac}}$ *in the* CKS-light *security model. Then there exists a factoring adversary $\mathcal{C}$ with:*

$$\mathrm{Adv}^{\text{CKS-light}}_{\mathcal{A},\mathrm{NIKE}_{\mathrm{fac}}}(k) \leq \mathrm{Adv}^{\mathrm{fac}}_{\mathcal{C},\mathtt{RSAgen}}(k) + O(2^{-\delta n(k)}).$$

The proof of Theorem 3 is given in Appendix C.

4.2 Towards a factoring-based scheme in the standard model

The security proof of $\mathrm{NIKE}_{\mathrm{fac}}$ above crucially uses the statistical properties of the random oracle H. If we accept an *interactive* key registration, we can however give a factoring-based NIKE scheme in the standard model. The basis of this scheme is the factoring-based IND-CCA secure encryption scheme of Hofheinz and Kiltz [26]. However, in adapting their scheme to the NIKE setting, we will have to find a way to *simultaneously* cope with two challenge ciphertexts (which correspond to the public keys of the challenge identities). To cope with this modified setting, we will set up a simulation that is able to decrypt all but *two* ciphertexts (resp. NIKE public keys).

In our description, let $\mathtt{RSAgen}$ as before, let $\mathrm{ChamH} : \{0,1\}^* \times \mathcal{R}_{\mathrm{Cham}} \to \mathbb{Z}_{2^k}$ be a chameleon hash function (see also Appendix A). Now consider the following scheme $\mathrm{NIKE}_{\text{fac-int}}$:

$\mathtt{CommonSetup}(1^k)$	$\mathtt{NIKE.KeyGen}(params, \mathsf{ID})$
$(N,P,Q) \xleftarrow{\$} \mathtt{RSAgen(1^k)}$	$x \xleftarrow{\$} \mathbb{Z}_{\lfloor N/4 \rfloor}$; $r \xleftarrow{\$} \mathcal{R}_{\mathrm{Cham}}$
$g, u_0, u_1, u_2 \xleftarrow{\$} \mathbb{QR}_N^+$,	$Z \leftarrow g^{x \cdot 2^{3k}}$;
where $\langle g \rangle = \mathbb{QR}_N^+$	$t \leftarrow \mathrm{ChamH}_{\mathsf{hk}}(Z \Vert \mathsf{ID}; r)$
$\mathsf{hk}, \mathsf{ck} \xleftarrow{\$} \mathtt{Cham.KeyGen}(1^k)$	$Y \leftarrow u_0 u_1^t {u_2}^{t^2}$; $X \leftarrow Y^x$
$params \leftarrow (N, g, u_0, u_1, u_2, \mathsf{hk})$	$pk \leftarrow (Z, X, r)$; $sk \leftarrow x$
Return $params$	Return (pk, sk)

$\mathtt{SharedKey}(\mathsf{ID}_1, pk_1, \mathsf{ID}_2, sk_2)$
If $(\mathsf{ID}_1 = \mathsf{ID}_2)$ or $pk_1 \notin \mathbb{QR}_N^+ \times \mathbb{QR}_N^+ \times \mathcal{R}_{\mathrm{Cham}}$ or $sk_2 \notin \mathbb{Z}_{\lfloor N/4 \rfloor}$ return $\perp$
Parse $pk_1 =: (Z_1, X_1, r_1)$ and $sk_2 =: x_2$
Return $\mathrm{BBS}_N(Z_1^{x_2 \cdot 2^{2k}})$

Note that correctness of the scheme follows from $Z_1^{x_2 \cdot 2^{2k}} = g^{x_1 \cdot x_2 \cdot 2^{5k}} = Z_2^{x_1 \cdot 2^{2k}}$. To prove security, we need to rely on the *consistency* of public keys. Concretely, the security reduction we will give can only authentically answer corrupt reveal queries for corrupt user keys $pk = (Z, X, r)$ that satisfy $Z = g^{x \cdot 2^{3k}}$, $X = (u_0 u_1^t u_2^{t^2})^x$ for $t = \mathrm{ChamH}_{\mathsf{hk}}(Z||\mathsf{ID}; r)$ and some x. Unlike in our upcoming pairing-based scheme, this kind of consistency is not (obviously) efficiently verifiable. Hence, the key registration process must ensure that only consistent user keys are registered, e.g., by having the user prove consistency in zero-knowledge (interactively, using x as witness).

On top of assuming consistent keys, we will also have to make an assumption about the distribution of (or rather, the ability to generate) primes. Namely, we will need to assume a PPT algorithm `PrimeGen` that, on input a $2k$-bit prime ρ, outputs a prime α such that $\alpha \bmod \rho$ has statistical distance $O(2^{-k})$ from the uniform distribution over $\mathbb{Z}_\rho$. Such an algorithm `PrimeGen` exists. This is an easy consequence of Dirichlet's theorem on the distribution of primes in arithmetic progressions: our generator simply samples integers of the form $\alpha_0 + i \cdot \rho$ for uniformly chosen $\alpha_0 \in \mathbb{Z}_\rho$ and $i = 1, 2, \ldots$, and checks them for primality. This algorithm can be rigorously proven to be efficient under the Generalized Riemann Hypothesis.

Theorem 4. *Under the factoring assumption relative to* `RSAgen`*, given an algorithm* `PrimeGen` *as above, and assuming that the chameleon hash function ChamH is collision-resistant, the scheme* $\mathrm{NIKE}_{\text{fac-int}}$ *is secure against all adversaries that only register consistent (in the sense above) user keys. In particular, suppose $\mathcal{A}$ is such an adversary against* $\mathrm{NIKE}_{\text{fac}}$ *in the* CKS-light *security model. Then there exists a BBS distinguisher $\mathcal{B}$ and a collision-finder $\mathcal{A}_{\mathcal{CH}}$ with:*

$$\mathrm{Adv}^{\text{CKS-light}}_{\mathcal{A},\mathrm{NIKE}_{\text{fac-int}}}(k) \le \mathrm{Adv}^{\mathrm{BBS}}_{\mathcal{B},\texttt{RSAgen}}(k) + \mathrm{Adv}^{\mathrm{coll}}_{\mathcal{A}_{\mathcal{CH}},\mathrm{ChamH}}(k) + O(2^{-k}). \tag{1}$$

The proof of Theorem 4 is given in Appendix D.

4.3 A Construction in the Standard Model from Pairings

We specify how to build a NIKE scheme, $\mathrm{NIKE}_{\text{dbdh-2}}$, that is secure in the *CKS-light* security model under the DBDH-2 assumption in the standard model. Our construction makes use of a tuple $\mathcal{PG}2 = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2, p, e, \psi)$, output by a parameter generator $\mathcal{G}2$, and a chameleon hash function $\mathrm{ChamH} : \{0,1\}^* \times \mathcal{R}_{\mathrm{Cham}} \to \mathbb{Z}_p$. This can be instantiated efficiently using the discrete-log based construction from [28] (see Appendix A for further details of chameleon hash functions). The component algorithms of the scheme $\mathrm{NIKE}_{\text{dbdh-2}}$ are defined as follows:

`CommonSetup`(1^k)
- $\mathcal{PG}2 \xleftarrow{\$} \mathcal{G}2(1^k)$,
- where $\mathcal{PG}2 = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2, p, e, \psi)$
- $u_0, u_1, u_2, S \xleftarrow{\$} \mathbb{G}_1^*$
- $\mathsf{hk}, \mathsf{ck} \xleftarrow{\$}$ `Cham.KeyGen`(1^k)
- $params \leftarrow (\mathcal{PG}2, u_0, u_1, u_2, S, \mathsf{hk})$
- Return $params$

`NIKE.KeyGen`$(params, \mathsf{ID})$
- $x \xleftarrow{\$} \mathbb{Z}_p$; $r \xleftarrow{\$} \mathcal{R}_{\mathrm{Cham}}$
- $Z \leftarrow g_2^x$;
- $t \leftarrow \mathrm{ChamH}_{\mathsf{hk}}(Z||\mathsf{ID}; r)$;
- $Y \leftarrow u_0 u_1^t u_2^{t^2}$; $X \leftarrow Y^x$
- $pk \leftarrow (X, Z, r)$; $sk \leftarrow x$
- Return (pk, sk)

`SharedKey`$(\mathsf{ID}_1, pk_1, \mathsf{ID}_2, sk_2)$
- If $\mathsf{ID}_1 = \mathsf{ID}_2$ return $\perp$
- Parse pk_1 as (X_1, Z_1, r_1) and sk_2 as x_2
- $t_1 \leftarrow \mathrm{ChamH}_{\mathsf{hk}}(Z_1||\mathsf{ID}_1; r_1)$
- If $e(X_1, g_2) \neq e(u_0 u_1^{t_1} u_2^{t_1^2}, Z_1)$
  - then $K_{1,2} \leftarrow \perp$
  - else $K_{1,2} \leftarrow e(S^{x_2}, Z_1)$
- Return $K_{1,2}$

The check in the `SharedKey` algorithm for valid public keys can be implemented by evaluating the bilinear map twice. It is clear that `SharedKey` defined in this way satisfies the requirement that entities ID_1 and ID_2 are able to compute a common key. To see this, note that $e(S^{x_2}, Z_1) = e(S, g_2)^{x_1, x_2}$. The identity space for this construction, $\mathcal{IDS}$, is $\{0,1\}^*$, while the space of shared keys is $\mathcal{SHK} = \mathbb{G}_T$. Public keys and parameters are compact. For example, at the 128-bit security level, using BN curves [29] and point compression, public keys consist of 768 bits plus an element from $\mathcal{R}_{\mathrm{Cham}}$.

As stated before, we can prove the above NIKE scheme to be secure under the DBDH-2 assumption in the sense of the *CKS-light* security model. Interestingly, our scheme can be generalised to use any *weak* $(2, \text{poly})$-PHF [21] in combination with a chameleon hash function. That is, Y (in the `NIKE.KeyGen` algorithm) would be the output of the *weak* $(2, \text{poly})$-PHF on input t, where t is the output of the chameleon hash function. We have given a specific construction here because suitable weak PHFs are currently rare. A further generalisation of our scheme could use any randomised $(2, \text{poly})$-PHF and avoid the chameleon hash, but no constructions for these are currently known.

Theorem 5. *Assume* ChamH *is a family of chameleon hash functions. Then* $\text{NIKE}_{\text{dbdh-2}}$ *is secure under the DBDH-2 assumption relative to generator* $\mathcal{G}2$*. In particular, suppose* $\mathcal{A}$ *is an adversary against* $\text{NIKE}_{\text{dbdh-2}}$ *in the* CKS-light *security model. Then there exists a DBDH-2 adversary* $\mathcal{B}$ *with:*

$$\text{Adv}^{\text{dbdh-2}}_{\mathcal{B},\mathcal{G}2}(k) \geq \text{Adv}^{\text{CKS-light}}_{\mathcal{A},\text{NIKE}_{\text{dbdh-2}}}(k) - \text{Adv}^{\text{coll}}_{\mathcal{A}_{\mathcal{CH}},\text{ChamH}}(k).$$

For the proof, see Appendix E.

5 From Non-interactive Key Exchange to Public Key Encryption

We give a conversion that takes a NIKE scheme that is secure in the *CKS-light* security model plus a strongly one-time secure signature scheme, and produces from it a KEM that is IND-CCA secure. From such a KEM, it is easy to construct an IND-CCA secure public key encryption scheme [30].

The formal definitions of KEM and OTS and their security can be found in Appendix A.

5.1 The Conversion from NIKE to KEM

We now present our conversion from a NIKE scheme to a KEM. For a NIKE scheme `NIKE` and an OTS scheme `OTS`, we construct a KEM `KEM(NIKE, OTS)` with the following algorithms:

- $\texttt{KEM.KeyGen}(1^k)$: This algorithm runs the algorithm $\texttt{CommonSetup}(1^k)$ of `NIKE` to obtain a set of system parameters, *params*. Then it picks $\mathsf{ID} \in \mathcal{IDS}$ uniformly and runs $\texttt{NIKE.KeyGen}(params, \mathsf{ID})$ to obtain a key pair (pk, sk). It sets $pk_{\texttt{KEM}} = (params, \mathsf{ID}, pk)$ and $sk_{\texttt{KEM}} = (\mathsf{ID}, sk)$.
- $\texttt{Enc}(pk_{\texttt{KEM}})$: This algorithm parses $pk_{\texttt{KEM}}$ as $(params, \mathsf{ID}, pk)$, runs `OTSKeyGen` to obtain a pair $(vk, sigk)$. This is repeated until $vk \neq \mathsf{ID}$. Next, it runs $\texttt{NIKE.KeyGen}(params, \mathsf{ID}' = vk)$ of `NIKE` to obtain a key pair (pk', sk') and runs $\texttt{OTSSign}(sigk, pk')$ to obtain σ, a signature on pk'. It then runs $\texttt{SharedKey}(\mathsf{ID}, pk, \mathsf{ID}' = vk, sk')$ of scheme `NIKE` to obtain a key $K \in \mathcal{SHK}$. The output is $(K, C = (vk, pk', \sigma))$.
- $\texttt{Dec}(sk_{\texttt{KEM}}, C)$: This algorithm first parses C as (vk, pk', σ) and $sk_{\texttt{KEM}}$ as (ID, sk). Next, it runs $\texttt{OTSVfy}(vk, pk', \sigma)$ and returns $\perp$ if the output is `reject` or if $vk = \mathsf{ID}$. Otherwise, it runs $\texttt{SharedKey}(\mathsf{ID}' = vk, pk', \mathsf{ID}, sk)$ and outputs the result, which may be $\perp$.

Notice that the ciphertexts in this scheme consist of a verification key from the OTS scheme, a public key from the NIKE scheme, and a one-time signature, while the encapsulated keys are elements of $\mathcal{SHK}$. As our next result shows, the resulting KEM is automatically IND-CCA secure if the NIKE scheme is secure in the *CKS-light* security model.

Theorem 6. *Suppose the NIKE scheme* `NIKE` *is secure in the* CKS-light *security model and* `OTS` *is a strongly secure one-time signature scheme. Then* `KEM(NIKE, OTS)` *is an IND-CCA secure KEM. More precisely, for any adversary* $\mathcal{A}$ *against* `KEM(NIKE, OTS)`*, there exists an adversary* $\mathcal{B}$ *against* `NIKE` *in the* CKS-light *security model or an adversary* $\mathcal{C}$ *against* `OTS` *having the same advantage. Moreover, if* $\mathcal{A}$ *makes* q_D *decapsulation queries, then* $\mathcal{B}$ *makes* q_D *register corrupt user queries and* q_D *corrupt reveal queries, while* $\mathcal{B}$*'s running time is roughly the same as that of* $\mathcal{A}$*.*

For the proof, see Appendix F.

Applying the above construction to the pairing-based NIKE scheme from the previous section results in an IND-CCA secure KEM with public keys (ID, pk) that consist of an identity string, two group elements (one in $\mathbb{G}_1$ and one in $\mathbb{G}_2$), and a key for the Chameleon hash function. Ciphertexts are slightly longer, containing in addition a verification key and a signature from the one-time signature scheme[7].

[7] Arguably, one might also include the public parameters *params* when evaluating the public key size.

6 Conclusions and Open Problems

We provided different security models for NIKE and explored the relationships between them. We then gave constructions for secure NIKE in the ROM and in the standard model. We also studied the relationship between NIKE and PKE, showing that a secure NIKE implies an IND-CCA secure PKE scheme.

There are several interesting open problems that arise from our work. One is to construct pairing-free NIKE schemes in the standard model. A challenge to doing so is that our pairing-based construction uses the pairing in a fundamental way in order to provide a publicly computable check on the validity of public keys. The RSA/factoring setting seems particularly challenging in this respect – we recall that our standard model, factoring-based scheme required that the adversary only register valid public keys, a condition that could be enforced in practice by having an interactive key registration protocol and insisting on proofs of validity during that protocol. Clearly, it is desirable from both a practical and a theoretical perspective to obtain schemes that are secure in the plain setting, where no such protocol is required.

Another open problem is to construct ID-based NIKE schemes that are provably secure in the standard model, moving beyond the ROM schemes analysed in [8,10]. Starting with known IBE schemes may be profitable, but the fact that these generally have randomised private key generation algorithms seems to make it hard to work backwards from IBE to ID-based NIKE.

Finally, it would be interesting to consider three-party NIKE schemes based on Joux's protocol [31]. Currently, there is no security model for such schemes, and no constructions which can handle adversarially-generated public keys.

References

1. Diffie, W., Hellman, M.E.: New directions in cryptography. IEEE Transactions on Information Theory **22**(6) (1976) 644–654
2. Çağatay Çapar, Goeckel, D., Paterson, K.G., Quaglia, E.A., Towsley, D., Zafer, M.: Signal-flow-based analysis of wireless security protocols. Information and Computation (to appear)
3. Dodis, Y., Katz, J., Smith, A., Walfish, S.: Composability and on-line deniability of authentication. In Reingold, O., ed.: TCC. Volume 5444 of Lecture Notes in Computer Science., Springer (2009) 146–162
4. Boyd, C., Mao, W., Paterson, K.G.: Key agreement using statically keyed authenticators. In Jakobsson, M., Yung, M., Zhou, J., eds.: ACNS. Volume 3089 of Lecture Notes in Computer Science., Springer (2004) 248–262
5. Jakobsson, M., Sako, K., Impagliazzo, R.: Designated verifier proofs and their applications. In Maurer, U.M., ed.: EUROCRYPT. Volume 1070 of Lecture Notes in Computer Science., Springer (1996) 143–154
6. Cash, D., Kiltz, E., Shoup, V.: The twin Diffie-Hellman problem and applications. In Smart, N.P., ed.: EUROCRYPT. Volume 4965 of Lecture Notes in Computer Science., Springer (2008) 127–145
7. Sakai, R., Ohgishi, K., Kasahara, M.: Cryptosystems based on pairing. In: The 2000 Symposium on Cryptography and Information Security. (2000) 26–28
8. Dupont, R., Enge, A.: Provably secure non-interactive key distribution based on pairings. Discrete Applied Mathematics **154**(2) (2006) 270–276
9. Gennaro, R., Halevi, S., Krawczyk, H., Rabin, T., Reidt, S., Wolthusen, S.D.: Strongly-resilient and non-interactive hierarchical key-agreement in MANETs. In Jajodia, S., López, J., eds.: ESORICS. Volume 5283 of Lecture Notes in Computer Science., Springer (2008) 49–65
10. Paterson, K.G., Srinivasan, S.: On the relations between non-interactive key distribution, identity-based encryption and trapdoor discrete log groups. Des. Codes Cryptography **52**(2) (2009) 219–241
11. Bellare, M., Rogaway, P.: Entity authentication and key distribution. In Stinson, D.R., ed.: CRYPTO. Volume 773 of Lecture Notes in Computer Science., Springer (1993) 232–249
12. Shoup, V.: On formal models for secure key exchange (version 4) (1999)
13. Bellare, M., Pointcheval, D., Rogaway, P.: Authenticated key exchange secure against dictionary attacks. In Preneel, B., ed.: EUROCRYPT. Volume 1807 of Lecture Notes in Computer Science., Springer (2000) 139–155
14. Canetti, R., Krawczyk, H.: Analysis of key-exchange protocols and their use for building secure channels. In Pfitzmann, B., ed.: EUROCRYPT. Volume 2045 of Lecture Notes in Computer Science., Springer (2001) 453–474
15. Krawczyk, H.: HMQV: A high-performance secure Diffie-Hellman protocol. In Shoup, V., ed.: CRYPTO. Volume 3621 of Lecture Notes in Computer Science., Springer (2005) 546–566
16. Bellare, M., Neven, G.: Multi-signatures in the plain public-key model and a general forking lemma. In Juels, A., Wright, R.N., di Vimercati, S.D.C., eds.: ACM Conference on Computer and Communications Security, ACM (2006) 390–399

17. Ristenpart, T., Yilek, S.: The power of proofs-of-possession: Securing multiparty signatures against rogue-key attacks. In Naor, M., ed.: EUROCRYPT. Volume 4515 of Lecture Notes in Computer Science., Springer (2007) 228–245
18. Boldyreva, A.: Threshold signatures, multisignatures and blind signatures based on the gap-Diffie-Hellman-group signature scheme. In Desmedt, Y., ed.: Public Key Cryptography. Volume 2567 of Lecture Notes in Computer Science., Springer (2003) 31–46
19. Lu, S., Ostrovsky, R., Sahai, A., Shacham, H., Waters, B.: Sequential aggregate signatures and multisignatures without random oracles. In Vaudenay, S., ed.: EUROCRYPT. Volume 4004 of Lecture Notes in Computer Science., Springer (2006) 465–485
20. Hofheinz, D., Kiltz, E.: The group of signed quadratic residues and applications. In Halevi, S., ed.: CRYPTO. Volume 5677 of Lecture Notes in Computer Science., Springer (2009) 637–653
21. Hofheinz, D., Jager, T., Kiltz, E.: Short signatures from weaker assumptions. In Lee, D.H., Wang, X., eds.: ASIACRYPT. Volume 7073 of Lecture Notes in Computer Science., Springer (2011) 647–666
22. Chatterjee, S., Sarkar, P.: Generalization of the selective-ID security model for HIBE protocols. In Yung, M., Dodis, Y., Kiayias, A., Malkin, T., eds.: Public Key Cryptography. Volume 3958 of Lecture Notes in Computer Science., Springer (2006) 241–256
23. ElGamal, T.: A public key cryptosystem and a signature scheme based on discrete logarithms. IEEE Transactions on Information Theory **31**(4) (1985) 469–472
24. Goldwasser, S., Micali, S., Rackoff, C.: The knowledge complexity of interactive proof systems. SIAM Journal on Computing **18**(1) (1989) 186–208
25. Fischlin, R., Schnorr, C.P.: Stronger security proofs for RSA and Rabin bits. Journal of Cryptology **13**(2) (2000) 221–244
26. Hofheinz, D., Kiltz, E.: Practical chosen ciphertext secure encryption from factoring. In Joux, A., ed.: EUROCRYPT. Volume 5479 of Lecture Notes in Computer Science., Springer (2009) 313–332
27. Galindo, D.: Boneh-Franklin identity based encryption revisited. In Caires, L., Italiano, G.F., Monteiro, L., Palamidessi, C., Yung, M., eds.: ICALP. Volume 3580 of Lecture Notes in Computer Science., Springer (2005) 791–802
28. Krawczyk, H., Rabin, T.: Chameleon signatures. In: NDSS. (2000)
29. Barreto, P.S.L.M., Naehrig, M.: Pairing-friendly elliptic curves of prime order. In Preneel, B., Tavares, S.E., eds.: Selected Areas in Cryptography. Volume 3897 of Lecture Notes in Computer Science., Springer (2005) 319–331
30. Cramer, R., Shoup, V.: Design and analysis of practical public-key encryption schemes secure against adaptive chosen ciphertext attack. SIAM Journal on Computing **33** (2003) 167–226
31. Joux, A.: A one round protocol for tripartite Diffie-Hellman. In Bosma, W., ed.: ANTS. Volume 1838 of Lecture Notes in Computer Science., Springer (2000) 385–394
32. Rogaway, P., Shrimpton, T.: Cryptographic hash-function basics: Definitions, implications, and separations for preimage resistance, second-preimage resistance, and collision resistance. In Roy, B.K., Meier, W., eds.: FSE. Volume 3017 of Lecture Notes in Computer Science., Springer (2004) 371–388
33. Shoup, V.: OAEP reconsidered. J. Cryptology **15**(4) (2002) 223–249

A Basic Definitions

A.1 Collision Resistant Hash Functions

Let $\mathrm{CRF} : \mathcal{F} \times \mathcal{M} \to \mathcal{Y}$ be a family of keyed-hash functions and let $\mathcal{A}_{\mathcal{H}}$ be an adversary. CRF is said to be *collision resistant* if, for a hash function $\mathrm{CRF}_f \in \mathrm{CRF}$ (where the hash key f is chosen at random from $\mathcal{F}$), it is infeasible for any efficient adversary $\mathcal{A}_{\mathcal{H}}$ to find two distinct values m and m' such that $\mathrm{CRF}_f(m) = \mathrm{CRF}_f(m')$. More formally, following [32], we define

$$\begin{aligned}\mathrm{Adv}^{\mathrm{coll}}_{\mathcal{A}_{\mathcal{H}},\mathrm{CRH}}(k) = \ & |\Pr[f \xleftarrow{\$} \mathcal{F}; (m, m') \xleftarrow{\$} \mathcal{A}_{\mathcal{H}}(f) : \\ & (m \neq m') \wedge (\mathrm{CRF}_f(m) = \mathrm{CRF}_f(m'))]|.\end{aligned}$$

The hash function family is said to be *collision resistant* if $\mathrm{Adv}^{\mathrm{coll}}_{\mathcal{A}_{\mathcal{H}},\mathrm{CRH}}$ is negligible in k for any polynomial-time adversary $\mathcal{A}_{\mathcal{H}}$.

A.2 Target Collision Resistant Hash Functions

The difference between a target collision resistant hash function TCR_f and a collision resistant hash function CRF_f is that, in the former case, it is infeasible for an adversary, given a value m, to find a distinct value m' such that $\mathrm{TCR}_f(m) = \mathrm{TCR}_f(m')$. More formally, we define

$$\begin{aligned}\mathrm{Adv}^{\mathrm{coll}}_{\mathcal{A}_{\mathcal{H}},\mathrm{TCR}}(k) = \ & |\Pr[f \xleftarrow{\$} \mathcal{F}, m \xleftarrow{\$} \mathcal{M};\ m' \leftarrow \mathcal{A}_{\mathcal{H}}(f,m) : \\ & (m \neq m') \wedge (\mathrm{TCR}_f(m) = \mathrm{TCR}_f(m'))]|.\end{aligned}$$

The hash function is said to be *target collision resistant* if $\mathrm{Adv}^{\mathrm{coll}}_{\mathcal{A}_{\mathcal{H}},\mathrm{TCR}}$ is negligible in k for any polynomial-time adversary $\mathcal{A}_{\mathcal{H}}$.

A.3 Chameleon Hash Functions

Chameleon hash functions [28] can be thought of as collision resistant hash functions with a trapdoor for finding collisions. Let k be a security parameter. A chameleon hash function ChamH : $\mathcal{D}\times\mathcal{R}_{\mathrm{Cham}} \to \mathcal{I}$, where $\mathcal{D}$ is the domain, $\mathcal{R}_{\mathrm{Cham}}$ the randomness space and $\mathcal{I}$ the range, is associated with a pair of public and private keys (the latter called a trapdoor). These keys are denoted respectively by hk and ck and are generated by a PPT algorithm $\mathtt{Cham.KeyGen}(1^k)$. The public key hk defines a chameleon hash function, denoted $\mathrm{ChamH}_{\mathsf{hk}}(\cdot,\cdot)$. On input a message m and a random string r, this function generates a hash value $\mathrm{ChamH}_{\mathsf{hk}}(m,r)$ which satisfies the following properties:
Collision resistance There is no efficient algorithm that on input the public key hk can find pairs m_1, r_1 and m_2, r_2 where $m_1 \neq m_2$ such that $\mathrm{ChamH}_{\mathsf{hk}}(m_1,r_1) = \mathrm{ChamH}_{\mathsf{hk}}(m_2,r_2)$, except with negligible probability in k.

Trapdoor collisions There is an efficient algorithm that on input the secret key ck, any pair m_1, r_1 and any additional message m_2, finds a value r_2 such that $\mathrm{ChamH}_{\mathsf{hk}}(m_1,r_1) = \mathrm{ChamH}_{\mathsf{hk}}(m_2,r_2)$. Also, for uniformly and independently chosen m_1, r_1 and m_2, r_2 is independently and uniformly distributed over $\mathcal{R}_{\mathrm{Cham}}$.

Uniformity All mesages m induce the same probability distribution on $\mathrm{ChamH}_{\mathsf{hk}}(m,r)$ for r chosen uniformly at random. This property prevents a third party, examining the value hash, from deducing any information about the hashed message.

More formally, we define the advantage of an adversary $\mathcal{A}_{\mathcal{CH}}$ against ChamH as

$$\begin{aligned}\mathrm{Adv}^{\mathrm{coll}}_{\mathcal{A}_{\mathcal{CH}},\mathrm{ChamH}}(k) = & |\Pr[\mathsf{hk} \xleftarrow{\$} \mathtt{Cham.KeyGen}(1^k); (m_1,r_1,m_2,r_2) \xleftarrow{\$} \mathcal{A}_{\mathcal{CH}}(\mathsf{hk}) : \\ & (m_1 \neq m_2) \wedge (\mathrm{ChamH}_{\mathsf{hk}}(m_1,r_1) = \mathrm{ChamH}_{\mathsf{hk}}(m_2,r_2))]|.\end{aligned}$$

The composition of a chameleon hash function and a (regular) collision resistant hash function (where the latter is applied first) results in a chameleon hash function.

A.4 KEMs and KEM Security

A *key encapsulation mechanism* $\mathtt{KEM} = (\mathtt{KEM.KeyGen}, \mathtt{Enc}, \mathtt{Dec})$ consists of three algorithms:

- $\mathtt{KEM.KeyGen}$, a probabilistic polynomial-time *key generation algorithm* that on input 1^k, outputs a public key/secret key pair (pk, sk).
- $\mathtt{Enc}$, a probabilistic polynomial-time *encapsulation algorithm* that takes as input a public key pk and outputs a symmetric key $K \in \mathcal{K}$, where $\mathcal{K}$ is the symmetric key space of the KEM, and a ciphertext C.
- $\mathtt{Dec}$, a deterministic polynomial-time *decapsulation algorithm* that takes as input a secret key sk and a ciphertext C, and outputs either a key $K \in \mathcal{K}$ or a special symbol $\perp$.

For correctness we require that for all $k \in \mathbb{N}$ and all $(K, C) \leftarrow \mathtt{Enc}(pk)$, we have

$$\Pr[\mathtt{Dec}(sk, C) = K] = 1.$$

Chosen-ciphertext security for a KEM is defined in terms of the following IND-CCA experiment (from [26]), where an adversary $\mathcal{A}$ is allowed to adaptively query a decapsulation oracle with ciphertexts of its choice and obtain the corresponding keys.

Definition 1 (IND-CCA security of a KEM). *Let* $\texttt{KEM} = (\texttt{KEM.KeyGen}, \texttt{Enc}, \texttt{Dec})$ *be a key encapsulation mechanism. For any PPT algorithm $\mathcal{A}$, we define the following experiments:*

Experiment $\mathrm{Exp}_{\mathcal{A},\mathrm{KEM}}^{\mathrm{CCA-real}}(\mathrm{k})$	Experiment $\mathrm{Exp}_{\mathcal{A},\mathrm{KEM}}^{\mathrm{CCA-rand}}(\mathrm{k})$
$(pk, sk) \leftarrow \texttt{KEM.KeyGen}(1^k)$	$(pk, sk) \leftarrow \texttt{KEM.KeyGen}(1^k)$
	$K \xleftarrow{\$} \mathcal{K}$
$(K^*, C^*) \leftarrow \texttt{Enc}(pk)$	$(K^*, C^*) \leftarrow \texttt{Enc}(pk)$
Return $\mathcal{A}^{\texttt{Dec}(sk,\cdot)}(pk, K^*, C^*)$	*Return* $\mathcal{A}^{\texttt{Dec}(sk,\cdot)}(pk, K, C^*)$

In the above experiment, the decapsulation oracle $\texttt{Dec}(sk, \cdot)$, when queried with a ciphertext $C \neq C^*$, returns $K \leftarrow \texttt{Dec}(sk, C)$; ($\texttt{Dec}(sk, \cdot)$ ignores queries $C = C^*$. The advantage of $\mathcal{A}$ in breaking $\texttt{KEM}$'s IND-CCA security is defined to be:

$$\mathrm{Adv}_{\mathcal{A},\mathrm{KEM}}^{\mathrm{CCA}}(k, q_D) = \frac{1}{2}\left|\Pr[\mathrm{Exp}_{\mathcal{A},\mathrm{KEM}}^{\mathrm{CCA-real}}(k) = 1] - \Pr[\mathrm{Exp}_{\mathcal{A},\mathrm{KEM}}^{\mathrm{CCA-rand}}(k) = 1]\right|.$$

where q_D is a bound on the number of decapsulation queries made by $\mathcal{A}$. A KEM scheme is said to be IND-CCA secure if $\mathrm{Adv}_{\mathcal{A},\mathrm{KEM}}^{\mathrm{CCA}}(k, q_D)$ is negligible for all polynomial-time adversaries $\mathcal{A}$.

A.5 One-time signatures

A *one-time signature (OTS) scheme* $\texttt{OTS} = (\texttt{OTSKeyGen}, \texttt{OTSSign}, \texttt{OTSVfy})$ consists of three algorithms:

- $\texttt{OTSKeyGen}$, a probabilistic polynomial-time *key generation algorithm* that on input 1^k, outputs a verfication/signing key pair $(vk, sigk)$.
- $\texttt{OTSSign}$, a probabilistic polynomial-time *signing algorithm* that takes as input a signing key $sigk$ and a message m outputs a signature σ.
- $\texttt{OTSVfy}$, a deterministic polynomial-time *verification algorithm* that takes as input a verification key vk, a message m and a signature σ, and outputs either $\texttt{reject}$ or $\texttt{accept}$.

For correctness we require that for all $k \in \mathbb{N}$, all $(vk, sigk) \leftarrow \texttt{OTSKeyGen}(1^k)$, all messages m, and all $\sigma \leftarrow \texttt{OTSSign}(sigk, m)$, we have

$$\Pr[\texttt{OTSVfy}(vk, m, \sigma) = \texttt{accept}] = 1.$$

Strong security for an OTS scheme is defined in terms of the following experiment.

Definition 2. *Let* $\texttt{OTS} = (\texttt{OTSKeyGen}, \texttt{OTSSign}, \texttt{OTSVfy})$ *be an OTS. For any PPT algorithm $\mathcal{A}$, we define the following experiment:*

Experiment $\mathrm{Exp}_{\mathcal{A},\texttt{OTS}}^{\mathrm{OTS}}(\mathrm{k})$
$(vk, sigk) \leftarrow \texttt{OTSKeyGen}(1^k)$
$(m^*, \sigma^*) \leftarrow \mathcal{A}^{\texttt{OTSSign}(sigk,\cdot)}(vk)$
Return $(\texttt{OTSVfy}(vk, m^*, \sigma^*) = \texttt{accept})$ *and* $(m^*, \sigma^*) \neq (m, \sigma))$

In the above experiment, the signing oracle $\texttt{OTSSign}(sigk, \cdot)$ can only be queried once on some message m which results in a signature σ. The advantage of $\mathcal{A}$ in breaking $\texttt{OTS}$'s strong one-time security is defined to be:

$$\mathrm{Adv}_{\mathcal{A},\texttt{OTS}}(k) = \Pr[\mathrm{Exp}_{\mathcal{A},\texttt{OTS}}^{\mathrm{OTS}}(k) = 1].$$

An OTS scheme is said to be strongly secure if $\mathrm{Adv}_{\mathcal{A},\texttt{OTS}}(k)$ is negligible for all polynomial-time adversaries $\mathcal{A}$.

B Relationships between NIKE Security Models

We show that the four NIKE security models discussed in Section 2.2 are polynomially equivalent. Note that the proofs contain many parts in common, but we give them here in full detail for completeness.

Theorem 7 (CKS-heavy ⇔ m-CKS-heavy). *A NIKE scheme* NIKE *is secure in the* m-CKS-heavy *model if and only if it is secure in the* CKS-heavy *model. In more detail, for any adversary* $\mathcal{A}$ *against* NIKE *in the* m-CKS-heavy *model, there is an adversary* $\mathcal{B}$ *that breaks* NIKE *in the* CKS-heavy *model with*

$$\mathrm{Adv}_{\mathcal{B}}^{\text{CKS-heavy}}(k, q_H, q_C, q_E, q'_{HR}, q_{CR}) = \mathrm{Adv}_{\mathcal{A}}^{\text{m-CKS-heavy}}(k, q_H, q_C, q_E, q_{HR}, q_{CR}, q_T)/q_T,$$

where $q'_{HR} \le q_{HR} + q_T$*. Conversely, for any* $\mathcal{B}$ *against* NIKE *in the* CKS-heavy *model, there is an* $\mathcal{A}$ *that breaks* NIKE *in the* m-CKS-heavy *model with*

$$\mathrm{Adv}_{\mathcal{A}}^{\text{m-CKS-heavy}}(k, q_H, q_C, q_E, q_{HR}, q_{CR}, 1) = \mathrm{Adv}_{\mathcal{B}}^{\text{CKS-heavy}}(k, q_H, q_C, q_E, q_{HR}, q_{CR}).$$

Proof. Clearly, security in the sense of the *m-CKS-heavy* model implies security in the sense of the *CKS-heavy* model, since the latter model is a limited case of the former. Here we prove that if a NIKE scheme NIKE is secure in the *CKS-heavy* model, it is also secure in the *m-CKS-heavy* model. We use the *hybrid argument* technique to relate the model with a single *test* query (the *CKS-heavy*model) to a model allowing multiple *test* queries (the *m-CKS-heavy* model) for a fixed bit b. We assume that there exists an adversary $\mathcal{A}$ against NIKE in the *m-CKS-heavy* model with advantage $\mathrm{Adv}_{\mathcal{A}}^{\text{m-CKS-heavy}}(k, q_H, q_C, q_E, q_{HR}, q_{CR}, q_T) = |\Pr[\hat{b} = b] - 1/2|$. Without loss of generality, we will assume that the q_T *test* queries made by $\mathcal{A}$ are all distinct. We will also assume that there are no two test queries $(\mathsf{ID}_A, \mathsf{ID}_B)$ and $(\mathsf{ID}_B, \mathsf{ID}_A)$. This assumption is also without loss of generality by correctness of NIKE. We consider a sequence of indistinguishable games $G_0, G_1, \ldots, G_{q_T}$, all defined over the same probability space. Starting from the actual adversarial game G_0 (attack game with respect to an adversary $\mathcal{A}$ against NIKE in the *m-CKS-heavy* model), when $b = 1$ (that is, *test* queries will always be answered with random keys), we make slight modifications between successive games, in such a way that the adversary's view is still indistinguishable among the games. The last game, Game G_{q_T}, will be exactly like Game G_0, except that this time $\mathcal{A}$'s challenger will use $b = 0$ to answer $\mathcal{A}$'s *test* queries. Note that this means that $\mathcal{A}$ can distinguish games Game G_0 and Game G_{q_T} with advantage $\mathrm{Adv}_{\mathcal{A}}^{\text{m-CKS-heavy}}(k, q_H, q_C, q_E, q_{HR}, q_{CR}, q_T) = |\Pr[\mathcal{A}(G_0) = 1] - \Pr[\mathcal{A}(G_{q_T}) = 1]|$. We write $\mathcal{A}(G_i)$ to denote adversary $\mathcal{A}$ playing in game G_i. For every i, $0 \le i \le q_T$, we define a *hybrid variable* H^i where the first i elements are the actual shared keys associated to the corresponding users involved in the first i *test* queries, and the $q_T - i$ following elements are random keys. That is, $H^i = ({K_{(\cdot,\cdot)}}^{(1)}, \ldots, {K_{(\cdot,\cdot)}}^{(i)}, R^{(i+1)}, \ldots, R^{(q_T)})$, where $K^{(i)}_{(\cdot,\cdot)}$ denotes the paired key between the two identities involved in the i-st *test* query and $R^{(j)}$, $i + 1 \le j \le q_T$, represents random keys. $\mathcal{A}$'s challenger will use H^i to answer *test* queries in game Game G_i.

Game G_0. Define G_0 to be the original game as described in the *m-CKS-heavy security* model when $b = 1$.

Game G_i $(1 \le i \le q_T)$. This game is identical to game Game G_{i-1}, except that whenever $\mathcal{A}$ makes its i-th *test* query on a pair of identities, say ID_A and ID_B, $\mathcal{A}$'s challenger will return to $\mathcal{A}$ the actual shared key, $K_{(\mathsf{ID}_A,\mathsf{ID}_B)}$, between those identities. Note that Games G_i and G_{i+1} differ in only one single *test* query.

We now construct an adversary $\mathcal{B}$ against NIKE in the sense of the *CKS-heavy* model. $\mathcal{B}$ plays the *CKS-heavy* security game with challenger $\mathcal{C}$ and acts as a challenger for $\mathcal{A}$.

$\mathcal{C}$ takes as input the security parameter 1^k, runs algorithm `CommonSetup` of the NIKE scheme and gives $\mathcal{B}$ *params*. $\mathcal{C}$ then takes a random bit b and answers any oracle queries for $\mathcal{B}$ until $\mathcal{B}$ outputs a bit $\hat{b}$.

$\mathcal{B}$ chooses a uniformly random $i \in \{0, \ldots, q_T - 1\}$, and invokes $\mathcal{A}$ on the vector $\overline{H} = ({K_{(\cdot,\cdot)}}^{(1)}, \ldots, {K_{(\cdot,\cdot)}}^{(i)}, \alpha, R^{(i+2)}, \ldots, R^{(q_T)})$. That is, in order to answer the first i *test* queries, $\mathcal{B}$ makes *honest reveal* queries on the same pair of identities to $\mathcal{C}$, obtaining real shared keys. $\mathcal{B}$ then returns the shared keys to $\mathcal{A}$. Whenever $\mathcal{A}$ makes its $(i + 1)$-st *test* query, $\mathcal{B}$ will make the same *test* query to its challenger, receiving a value α. $\mathcal{B}$ gives α to $\mathcal{A}$. For all other *test* queries, $\mathcal{B}$ will respond with a random value. For all other queries made by $\mathcal{A}$, $\mathcal{B}$ passes these queries to $\mathcal{C}$ and then relays the responses to $\mathcal{A}$.

Now, if α is the actual key associated to the identities involved in the *test* query $test_{i+1}(\cdot,\cdot)$, then $\mathcal{A}$ was playing game G_{i+1}. Otherwise, if α is a random value, $\mathcal{A}$ was playing game G_i. Whenever $\mathcal{A}$ terminates by outputting a bit $\hat{b}$, then $\mathcal{B}$ outputs the same bit. Note that by our assumption the q_T *test* queries made by $\mathcal{A}$ are all distinct. So, $\mathcal{B}$ will never make an *honest reveal* query on the same pair of identities that will be involved in its *test* query. Note also that as explained before, the only difference between games G_i and G_{i+1} is a single *test* query.

Let G'_0 and G'_1 be the games played by $\mathcal{B}$ against `NIKE` in the *CKS-heavy* model when $b = 0$ and $b = 1$, respectively. We have:

$$\Pr[\mathcal{B}(G'_0) = 1] = \frac{1}{q_T} \sum_{i=0}^{q_T-1} \Pr[\mathcal{A}(G_{i+1}) = 1]$$

and

$$\Pr[\mathcal{B}(G'_1) = 1] = \frac{1}{q_T} \sum_{i=0}^{q_T-1} \Pr[\mathcal{A}(G_i) = 1].$$

It therefore follows that:

$$\begin{aligned}
&\mathrm{Adv}_{\mathcal{B}}^{\text{CKS-heavy}}(k, q_H, q_C, q_E, q'_{HR}, q_{CR}) = |\Pr[\mathcal{B}(G'_0) = 1] - \Pr[\mathcal{B}(G'_1) = 1]| = \\
&\frac{1}{q_T} \left| \sum_{i=0}^{q_T-1} \Pr[\mathcal{A}(G_{i+1}) = 1] - \sum_{i=0}^{q_T-1} \Pr[\mathcal{A}(G_i) = 1] \right| = \\
&\frac{1}{q_T} \left| \Pr[\mathcal{A}(G_0) = 1] - \Pr[\mathcal{A}(G_{q_T}) = 1] \right| = \\
&\mathrm{Adv}_{\mathcal{A}}^{\text{m-CKS-heavy}}(k, q_H, q_C, q_E, q_{HR}, q_{CR}, q_T)/q_T.
\end{aligned}$$

This concludes our proof. □

Theorem 8 (CKS-light ⇔ CKS). *A NIKE scheme* `NIKE` *is secure in the* CKS *model if and only if it is also secure in the* CKS-light *model. In more detail, for any adversary $\mathcal{A}$ against* `NIKE` *in the* CKS *model, there is an adversary $\mathcal{B}$ that breaks* `NIKE` *in the* CKS-light *model with*

$$\mathrm{Adv}_{\mathcal{B}}^{\text{CKS-light}}(k, q'_C, q'_{CR}) \geq 2\mathrm{Adv}_{\mathcal{A}}^{\text{CKS}}(k, q_H, q_C, q_{CR}, q_T)/{q_H}^2 q_T,$$

where $q'_C \leq q_C + q_H$ and $q'_{CR} \leq q_{CR}$. Conversely, for any $\mathcal{B}$ against `NIKE` *in the* CKS-light *model, there is an $\mathcal{A}$ that breaks* `NIKE` *in the* CKS *model with*

$$\mathrm{Adv}_{\mathcal{A}}^{\text{CKS}}(k, 2, q_C, q_{CR}, 1) = \mathrm{Adv}_{\mathcal{B}}^{\text{CKS-light}}(k, q_C, q_{CR}).$$

Proof. Clearly, security in the sense of the *CKS* model implies security in the sense of the *CKS-light* model. Here we prove that if a NIKE scheme `NIKE` is secure in the *CKS-light* model, then it is also secure in the *CKS* model. We assume that there exists an adversary $\mathcal{A}$ against `NIKE` in the *CKS* model with advantage $\mathrm{Adv}_{\mathcal{A}}^{\text{CKS}}(k, q_H, q_C, q_{CR}, q_T) = |\Pr[\hat{b} = b] - 1/2|$. We consider a sequence of games $G_0, G_1, \ldots, G_{q_T}$, all defined over the same probability space. Starting from the actual adversarial game G_0 (attack game with respect to an adversary $\mathcal{A}$ against `NIKE` in the *CKS* model), when $b = 1$ (that is, *test* queries will always be answered with random keys), we make slight modifications between successive games, in such a way that the adversary's view is still indistinguishable among the games. The last game, Game G_{q_T}, will be exactly like Game G_0, except that this time $\mathcal{A}$'s challenger will use $b = 0$ to answer $\mathcal{A}$'s *test* queries. Note that this means that $\mathcal{A}$ can distinguish games Game G_0 and Game G_{q_T} with advantage $\mathrm{Adv}_{\mathcal{A}}^{\text{CKS}}(k, q_H, q_C, q_{CR}, q_T) = |\Pr[\mathcal{A}(G_0) = 1] - \Pr[\mathcal{A}(G_{q_T}) = 1]|$. We write $\mathcal{A}(G_i)$ to denote adversary $\mathcal{A}$ playing in game G_i. For every i, $0 \leq i \leq q_T$, we define a *hybrid variable* H^i where the first i elements are the actual shared keys associated to the corresponding users involved in the first i *test* queries, and the $q_T - i$ following elements are random keys. That is, $H^i = ({K_{(\cdot,\cdot)}}^{(1)}, \ldots, {K_{(\cdot,\cdot)}}^{(i)}, R^{(i+1)}, \ldots, R^{(q_T)})$, where $K_{(\cdot,\cdot)}^{(i)}$ denotes the paired key between the two identities involved in the i-th *test* query and $R^{(j)}$, $i+1 \leq j \leq q_T$, represents random keys. $\mathcal{A}$'s challenger will use H^i to answer *test* queries in game Game G_i.

Game G_0. Define G_0 to be the original game as described in the *CKS* security model when $b = 1$.

Game G_i $(1 \le i \le q_T)$. This game is identical to game Game G_{i-1}, except that whenever $\mathcal{A}$ makes its i-th *test* query on a pair of identities, say ID_A and ID_B, $\mathcal{A}$'s challenger will return to $\mathcal{A}$ the actual shared key, $K_{(\mathsf{ID}_A,\mathsf{ID}_B)}$, between those identities. Note that Games G_i and G_{i+1} differ in only one single *test* query.

We now construct an adversary $\mathcal{B}$ against `NIKE` in the sense of the *CKS-light* model. $\mathcal{B}$ plays the *CKS-light* security game with challenger $\mathcal{C}$ and acts as a challenger for $\mathcal{A}$.

$\mathcal{C}$ takes as input the security parameter 1^k, runs algorithm `CommonSetup` of the NIKE scheme and gives $\mathcal{B}$ *params*. $\mathcal{C}$ then takes a random bit b and answers oracle queries for $\mathcal{B}$ until $\mathcal{B}$ outputs a bit $\hat{b}$.

Let q_T and q_H be bounds on the number of *test* queries and *register honest user* ID queries, respectively, made to $\mathcal{B}$ by $\mathcal{A}$ in the course of its attack. Without loss of generality, we assume that the q_T *test* queries are all distinct. $\mathcal{B}$ chooses a random $i \in \{0, \dots, q_T - 1\}$ and two distinct indices I and J uniformly at random from $\{1, 2, \dots, q_H\}$. Effectively $\mathcal{B}$ is guessing that the I-th and J-th identities to be honestly registered by $\mathcal{A}$ will be involved in the $(i+1)$-st *test* query made by $\mathcal{A}$.

$\mathcal{A}$ makes a series of queries which $\mathcal{B}$ answers as follows:

- *Register corrupt user ID*: If $\mathcal{A}$ makes a *register corrupt user ID* query, supplying (ID, pk), then $\mathcal{B}$ makes the same *register corrupt user ID* query to $\mathcal{C}$. $\mathcal{C}$ records the tuple $(corrupt, \mathsf{ID}, pk, \perp)$.
- *Register honest user ID*: Here $\mathcal{A}$ supplies a string ID to $\mathcal{B}$. If this is the I-th or J-th such query, then $\mathcal{B}$ makes the same *register honest user* ID query to $\mathcal{C}$, setting $\mathsf{ID}_I = \mathsf{ID}$ or $\mathsf{ID}_J = \mathsf{ID}$ as appropriate. On input *params* and ID, $\mathcal{C}$ runs `NIKE.KeyGen`, generating a key pair (pk, sk), records $(honest, \mathsf{ID}, pk, sk)$ and returns pk to $\mathcal{B}$. If $\mathsf{ID} \notin \{\mathsf{ID}_I, \mathsf{ID}_J\}$, then $\mathcal{B}$ generates a key pair (pk, sk), by running algorithm `NIKE.KeyGen` on input *params* and ID, and makes a *register corrupt user* ID query to $\mathcal{C}$ on inputs the string ID and the public key pk. $\mathcal{B}$ then gives pk to $\mathcal{A}$.
- *Corrupt reveal*: Whenever $\mathcal{A}$ supplies two identities $\mathsf{ID}, \mathsf{ID}'$, where ID was registered by $\mathcal{A}$ as corrupt and ID' was registered as honest, $\mathcal{B}$ will check if $\mathsf{ID}' \in \{\mathsf{ID}_I, \mathsf{ID}_J\}$. If so, $\mathcal{B}$ will make the same *corrupt reveal* query to $\mathcal{C}$, obtaining $K_{(\mathsf{ID},\mathsf{ID}')}$, and give the result to $\mathcal{A}$. If $\mathsf{ID}' \notin \{\mathsf{ID}_I, \mathsf{ID}_J\}$, $\mathcal{B}$ runs `SharedKey` on input $(\mathsf{ID}, pk_{\mathsf{ID}}, \mathsf{ID}', sk_{\mathsf{ID}'})$. Note that in this case, $\mathcal{B}$ has $sk_{\mathsf{ID}'}$ because it generated for itself the pair $(pk_{\mathsf{ID}'}, sk_{\mathsf{ID}'})$. $\mathcal{B}$ gives $K_{(\mathsf{ID}',\mathsf{ID}')}$ to $\mathcal{A}$.
- *Test*: $\mathcal{B}$ answers $\mathcal{A}$'s *test* queries according to the vector

$$\overline{H} = (K^{(1)}_{(\cdot,\cdot)}, \dots, K^{(i)}_{(\cdot,\cdot)}, \alpha, R^{(i+2)}, \dots, R^{(q_T)}).$$

That is, $\mathcal{B}$ will answer the first i *test* queries with the actual shared keys associated to the corresponding users involved in those *test* queries, the $(i+1)$-st *test* query with a value that can be either the actual shared key associated to the users involved in that *test* query or a random value, and the other $q_T - i - 1$ *test* queries with random values. Next, we explain in more detail exactly how $\mathcal{B}$ handles $\mathcal{A}$'s *test* queries.

When $\mathcal{A}$ makes its j-th $(j \le i)$ *test* query on a pair of identities $\{\mathsf{ID}, \mathsf{ID}'\}$, that were registered as honest users, $\mathcal{B}$ will check if $\{\mathsf{ID}, \mathsf{ID}'\} = \{\mathsf{ID}_I, \mathsf{ID}_J\}$. If so, $\mathcal{B}$ aborts the simulation. Otherwise, suppose $|\{\mathsf{ID}, \mathsf{ID}'\} \cap \{\mathsf{ID}_I, \mathsf{ID}_J\}| \le 1$. Then we consider 3 cases:

1. $\mathsf{ID} \cap \{\mathsf{ID}_I, \mathsf{ID}_J\} \neq \emptyset$ and $\mathsf{ID}' \cap \{\mathsf{ID}_I, \mathsf{ID}_J\} = \emptyset$

   This means that $\mathcal{B}$ registered ID' as corrupt user and ID as honest user. $\mathcal{B}$ runs `SharedKey` on input $(\mathsf{ID}, pk_{\mathsf{ID}}, \mathsf{ID}', sk_{\mathsf{ID}'})$. Note that $\mathcal{B}$ has $sk_{\mathsf{ID}'}$ because it generated for itself the pair $(pk_{\mathsf{ID}'}, sk_{\mathsf{ID}'})$. $\mathcal{B}$ gives $K_{(\mathsf{ID},\mathsf{ID}')}$ to $\mathcal{A}$.

2. $\mathsf{ID} \cap \{\mathsf{ID}_I, \mathsf{ID}_J\} = \emptyset$ and $\mathsf{ID}' \cap \{\mathsf{ID}_I, \mathsf{ID}_J\} \neq \emptyset$

   This means that $\mathcal{B}$ registered ID as corrupt user and ID' as honest user. $\mathcal{B}$ runs `SharedKey` on input $(\mathsf{ID}', pk_{\mathsf{ID}'}, \mathsf{ID}, sk_{\mathsf{ID}})$. Note that $\mathcal{B}$ has sk_{ID} because it generated for itself the pair $(pk_{\mathsf{ID}}, sk_{\mathsf{ID}})$. $\mathcal{B}$ gives $K_{(\mathsf{ID},\mathsf{ID}')}$ to $\mathcal{A}$.

3. $\{\mathsf{ID}, \mathsf{ID}'\} \cap \{\mathsf{ID}_I, \mathsf{ID}_J\} = \emptyset$

   Here both of the identities, ID and ID' were registered by $\mathcal{B}$ as corrupt users, so that $\mathcal{B}$ cannot make a *corrupt reveal* query on them. Instead $\mathcal{B}$ runs `SharedKey` on inputs $(\mathsf{ID}', pk_{\mathsf{ID}'}, \mathsf{ID}, sk_{\mathsf{ID}})$, and returns $K_{(\mathsf{ID},\mathsf{ID}')}$ to $\mathcal{A}$.

When $\mathcal{A}$ makes its $(i+1)$-st *test* query on a pair of identities $\{\mathsf{ID}, \mathsf{ID}'\}$, $\mathcal{B}$ checks if $\{\mathsf{ID}, \mathsf{ID}'\} = \{\mathsf{ID}_I, \mathsf{ID}_J\}$. If not, $\mathcal{B}$ aborts the simulation. If $\{\mathsf{ID}, \mathsf{ID}'\} = \{\mathsf{ID}_I, \mathsf{ID}_J\}$, $\mathcal{B}$ makes the same *test* query to $\mathcal{C}$ receiving α. $\mathcal{B}$ gives α to $\mathcal{A}$. For all other *test* queries $\mathcal{B}$ will respond with a random value.

Whenever $\mathcal{A}$ terminates by outputting a bit $\hat{b}$, then $\mathcal{B}$ outputs the same bit. Now, if α is the actual key $K_{(\mathsf{ID}_A, \mathsf{ID}_B)}$ associated to $(\mathsf{ID}_A, \mathsf{ID}_B)$ (the identities involved in the $(i+1)$-st *test* query made by $\mathcal{A}$), then $\mathcal{A}$ was playing game *Game* G_{i+1}. Otherwise, if α is a random value, $\mathcal{A}$ was playing game *Game* G_i.

We now assess $\mathcal{B}$'s success probability. Let G_0' and G_1' be the games played by $\mathcal{B}$ against `NIKE` in the *CKS-light* model when $b = 0$ and $b = 1$, respectively. Let F denote the event that $\mathcal{B}$ is not forced to abort during its simulation. It is easy to see that $\Pr(F) \geq 1/\binom{q_H}{2} \geq 2/{q_H}^2$.

We have:

$$\Pr[\mathcal{B}(G_0') = 1] = \Pr[F]\frac{1}{q_T}\sum_{i=0}^{q_T-1}\Pr[\mathcal{A}(G_{i+1}) = 1]$$

and

$$\Pr[\mathcal{B}(G_1') = 1] = \Pr[F]\frac{1}{q_T}\sum_{i=0}^{q_T-1}\Pr[\mathcal{A}(G_i) = 1]$$

It therefore follows that:

$$\begin{aligned}
\mathrm{Adv}_{\mathcal{B}}^{\text{CKS-light}}(k, q_C', q_{CR}') &= |\Pr[\mathcal{B}(G_0') = 1] - \Pr[\mathcal{B}(G_1') = 1]| \\
&= \frac{\Pr[F]}{q_T}\left|\sum_{i=0}^{q_T-1}\Pr[\mathcal{A}(G_{i+1}) = 1] - \sum_{i=0}^{q_T-1}\Pr[\mathcal{A}(G_i) = 1]\right| \\
&= \frac{\Pr[F]}{q_T}|\Pr[\mathcal{A}(G_0) = 1] - \Pr[\mathcal{A}(G_{q_T}) = 1]| \\
&= \frac{\Pr[F]}{q_T}\mathrm{Adv}_{\mathcal{A}}^{\text{CKS}}(k, q_H, q_C, q_{CR}, q_T) \\
&\geq 2\mathrm{Adv}_{\mathcal{A}}^{\text{CKS}}(k, q_H, q_C, q_{CR}, q_T)/{q_H}^2 q_T.
\end{aligned}$$

This concludes our proof. □

Theorem 9 (CKS-heavy ⇔ CKS-light). *A* NIKE *scheme* `NIKE` *is secure in the* CKS-heavy *model if and only if it is also secure in the* CKS-light *model. In more detail, for any adversary $\mathcal{A}$ against* `NIKE` *in the* CKS-heavy *model, there is an adversary $\mathcal{B}$ that breaks* `NIKE` *in the* CKS-light *model with*

$$\mathrm{Adv}_{\mathcal{B}}^{\text{CKS-light}}(k, q_C', q_{CR}') \geq 2\mathrm{Adv}_{\mathcal{A}}^{\text{CKS-heavy}}(k, q_H, q_C, q_E, q_{HR}, q_{CR})/{q_H}^2,$$

where $q_C' \leq q_C + q_H$ and $q_{CR}' \leq q_{CR}$. Conversely, for any $\mathcal{B}$ against `NIKE` *in the* CKS-light *model, there is an $\mathcal{A}$ that breaks* `NIKE` *in the* CKS-heavy *model with*

$$\mathrm{Adv}_{\mathcal{A}}^{\text{CKS-heavy}}(k, 2, q_C, 0, 0, q_{CR}) = \mathrm{Adv}_{\mathcal{B}}^{\text{CKS-light}}(k, q_C, q_{CR}).$$

Proof. Clearly, security in the sense of the *CKS-heavy* model implies security in the sense of the *CKS-light* model. Here we prove that if a NIKE scheme `NIKE` is secure in the *CKS-light* model, it is also secure in the *CKS-heavy* model. Suppose there is an adversary $\mathcal{A}$ against `NIKE` in the *CKS-heavy* model with advantage $\mathrm{Adv}_{\mathcal{A}}^{\text{CKS-heavy}}(k, q_H, q_C, q_E, q_{HR}, q_{CR})$. We show how to construct an algorithm $\mathcal{B}$ against `NIKE` in the *CKS-light* model that uses $\mathcal{A}$ to break `NIKE` with advantage $\mathrm{Adv}_{\mathcal{B}}^{\text{CKS-light}}(k, q_C', q_{CR}') \geq 2\mathrm{Adv}_{\mathcal{A}}^{\text{CKS-heavy}}(k, q_H, q_C, q_E, q_{HR}, q_{CR})/{q_H}^2$, where k is the security parameter.

$\mathcal{B}$ plays the *CKS-light* security game with challenger $\mathcal{C}$ and acts as a challenger for $\mathcal{A}$.

$\mathcal{C}$ takes as input the security parameter 1^k, runs algorithm `CommonSetup` of the NIKE scheme and gives $\mathcal{B}$ *params*. $\mathcal{C}$ then takes a random bit b and answers oracle queries for $\mathcal{B}$ until $\mathcal{B}$ outputs a bit $\hat{b}$.

Let q_H be a bound on the number of *register honest user ID* queries made to $\mathcal{B}$ by $\mathcal{A}$ in the course of its attack. $\mathcal{B}$ chooses two distinct indices I and J uniformly at random from $\{1,2,\ldots,q_H\}$.$\mathcal{A}$ makes a series of queries which $\mathcal{B}$ answers as follows:

– *Register corrupt user ID*: If $\mathcal{A}$ makes a *register corrupt user ID* query supplying (ID, pk) as input, $\mathcal{B}$ makes the same *register corrupt user ID* query to $\mathcal{C}$. $\mathcal{C}$ records the tuple $(\mathit{corrupt}, \mathsf{ID}, pk, \perp)$.

– *Register honest user ID*: Here $\mathcal{A}$ supplies a string ID to $\mathcal{B}$. If this is the I-th or J-th such query, then $\mathcal{B}$ sets $\mathsf{ID}_I = \mathsf{ID}$ or $\mathsf{ID}_J = \mathsf{ID}$ as appropriate. Then $\mathcal{B}$ makes the same *register honest user ID* query to $\mathcal{C}$. On input *params* and ID, $\mathcal{C}$ runs `NIKE.KeyGen`, generating a key pair (pk, sk), records $(honest, \mathsf{ID}, pk, sk)$ and returns pk to $\mathcal{B}$. $\mathcal{B}$ gives pk to $\mathcal{A}$. Otherwise, when this is not the I-th or J-th such query, $\mathcal{B}$ generates a key pair (pk, sk), by running algorithm `NIKE.KeyGen` on input *params* and ID, and makes a *register corrupt user ID* query to $\mathcal{C}$ on inputs the string ID and the public key pk. $\mathcal{B}$ then gives pk to $\mathcal{A}$.
– *Extract*: Whenever $\mathcal{A}$ makes an *extract* query on a user identity ID, that was registered by $\mathcal{A}$ as honest, $\mathcal{B}$ checks if $\mathsf{ID} \in \{\mathsf{ID}_I, \mathsf{ID}_J\}$. If so, $\mathcal{B}$ aborts the simulation. If $\mathsf{ID} \notin \{\mathsf{ID}_I, \mathsf{ID}_J\}$, $\mathcal{B}$ finds ID in the list $(honest, \mathsf{ID}, pk, sk)$ and returns sk to $\mathcal{A}$.
– *Honest reveal*: Whenever $\mathcal{A}$ supplies two identities $\mathsf{ID}, \mathsf{ID}'$, where ID and ID' were registered by $\mathcal{A}$ as honest users, $\mathcal{B}$ will check if $\{\mathsf{ID}, \mathsf{ID}'\} = \{\mathsf{ID}_I, \mathsf{ID}_J\}$. If so, $\mathcal{B}$ aborts the simulation. (Note that in this case $\mathcal{B}$ does not have either of the secret keys needed to compute the paired key between the two identities.) Otherwise, $\mathcal{B}$ runs `SharedKey` on the appropriate inputs. (Note that in this case, $\mathcal{B}$ has at least one of the secret keys needed to execute `SharedKey`.)
– *Corrupt reveal*: Now, if $\mathcal{A}$ supplies two identities $\mathsf{ID}, \mathsf{ID}'$, where ID was registered by $\mathcal{A}$ as corrupt and ID' was registered as honest, $\mathcal{B}$ will check if $\mathsf{ID}' \in \{\mathsf{ID}_I, \mathsf{ID}_J\}$. If so, $\mathcal{B}$ will make a *Corrupt reveal* query to $\mathcal{C}$ obtaining the shared key between ID and ID', $K_{(\mathsf{ID},\mathsf{ID}')}$. $\mathcal{B}$ then returns the result to $\mathcal{A}$. If $\mathsf{ID}' \notin \{\mathsf{ID}_I, \mathsf{ID}_J\}$, then this means that $\mathcal{B}$ has $sk_{\mathsf{ID}'}$. Then $\mathcal{B}$ runs `SharedKey` using $sk_{\mathsf{ID}'}$ as an input and returns $K_{(\mathsf{ID},\mathsf{ID}')}$ to $\mathcal{A}$.
– *Test*: Whenever $\mathcal{A}$ makes its *test* query on a pair of user identities $\{\mathsf{ID}_A, \mathsf{ID}_B\}$, $\mathcal{B}$ checks if $\{\mathsf{ID}_A, \mathsf{ID}_B\} = \{\mathsf{ID}_I, \mathsf{ID}_J\}$. If so, $\mathcal{B}$ makes a *test* query to $\mathcal{C}$ on $\{\mathsf{ID}_A, \mathsf{ID}_B\}$ and gives the result to $\mathcal{A}$. If not, $\mathcal{B}$ aborts simulation.

This completes our description of $\mathcal{B}$'s simulation. When $\mathcal{A}$ terminates by outputting a bit $\hat{b}$ then $\mathcal{B}$ outputs the same bit. We now assess $\mathcal{B}$'s success probability. Let F denote the event that $\mathcal{B}$ is not forced to abort during its simulation. It is easy to see that $\Pr(F) \geq 1/\binom{q_H}{2} \geq 2/{q_H}^2$. Thus, we see that:

$$\mathrm{Adv}_{\mathcal{B}}^{\text{CKS-light}}(k, q'_C, q'_{CR}) \geq 2\mathrm{Adv}_{\mathcal{A}}^{\text{CKS-heavy}}(k, q_H, q_C, q_E, q_{HR}, q_{CR})/{q_H}^2.$$

This concludes the proof. □

C Proof of Theorem 3

Proof. Suppose $\mathcal{A}$ is an adversary against $\mathrm{NIKE}_{\mathrm{fac}}$ in the *CKS-light* security model. We first show how to construct an adversary $\mathcal{B}$ that uses $\mathcal{A}$ to solve the Double Strong DH problem in the group of *Signed Quadratic Residues* ($\mathbb{QR}_N^+$), where N is generated by `RSAgen`, and then use Theorem 2 to construct a factoring adversary $\mathcal{C}$. $\mathcal{B}$'s input is $(N, g, X = g^x, Y = g^y)$, where g is a generator of $\mathbb{QR}_N^+$ and (g^x, g^y) is an instance of the CDH problem in $\mathbb{QR}_N^+$. $\mathcal{B}$'s task is to compute g^{xy}, given access to two decisional oracles $\mathrm{DDH}_{g,X}(\cdot,\cdot)$ and $\mathrm{DDH}_{g,Y}(\cdot,\cdot)$. $\mathcal{B}$ acts as a challenger for $\mathcal{A}$.

$\mathcal{B}$ gives $\mathcal{A}$ the tuple (H, N, g), where H is a random oracle controlled by $\mathcal{B}$. $\mathcal{B}$ maintains a list L, initially empty, to store random oracle responses or responses to paired keys. $\mathcal{A}$ makes a series of queries which $\mathcal{B}$ answers as follows.

– *Register honest user ID*: When $\mathcal{B}$ receives register honest user ID queries for identities A and B, $\mathcal{B}$ sets $pk_A = X$ and $pk_B = Y$.
– *Register corrupt user ID*: Here, $\mathcal{B}$ receives a public key pk and a string ID from $\mathcal{A}$, and registers them. As in the original attack game, $\mathcal{B}$ aborts if ID equals one of the honest identities, A or B.
– *Corrupt reveal queries*: In order to output the paired key between one of the two honest users, say A, and a corrupt user, say D, $\mathcal{B}$ checks if A and D already appears in an entry of the form (A, D, h, R) on the list L (without loss of generality, assume $A < D$). If so, then $\mathcal{B}$ returns $K_{A,D} = R$ in response to $\mathcal{A}$'s query. Otherwise, $\mathcal{B}$ replies with $R \xleftarrow{\$} \{0,1\}^k$ and adds $(A, D, \perp, R)$ to L. Notice that by setup $sk_A = x = \mathrm{dlog}_g X$ (unknown to $\mathcal{B}$), so the 'correct' key would be $K_{A,D} = H(A, D, pk_D^x)$. (We assume $pk_D \in \mathbb{QR}_N^+$, otherwise $\mathcal{B}$ returns $\perp$.)
– *Test query*: At some point during the simulation, $\mathcal{A}$ makes a single *Test query* on the pair of identities (A, B). $\mathcal{B}$ outputs a randomly generated value $R \in \{0,1\}^k$. Notice that the 'correct' key $K_{A,B}$ that would be computed by $\mathcal{B}$ in responding to this query is equal to $H(A, B, g^{xy})$ (w.l.o.g. assuming $A < B$).

– *H queries*: On input $(\mathsf{ID}_1, \mathsf{ID}_2, h)$, w.l.o.g. $\mathsf{ID}_1, \mathsf{ID}_2 \in \{0,1\}^*$, $\mathsf{ID}_1 < \mathsf{ID}_2$, and $h \in \mathbb{QR}_N^+$, $\mathcal{B}$ answers $\mathcal{A}$'s *H queries* as follows. $\mathcal{B}$ checks if $(\mathsf{ID}_1, \mathsf{ID}_2, h, R)$ is already on the list for some R; If so, it outputs R. Otherwise, $\mathcal{B}$ checks if an entry of the form $(\mathsf{ID}_1, \mathsf{ID}_2, \perp, R)$ is on the list for at least ID_1 or ID_2 equals one of the two honest identities A or B. So let $\mathsf{ID}_1 = A$ ($\mathsf{ID}_1 = B$), that is, $pk_1 = X$ ($pk_1 = Y$). If ID_2 is also a registered identity, then $\mathcal{B}$ checks if h is the 'correct' DH value for $(\mathsf{ID}_1, \mathsf{ID}_2)$ using one of the DDH oracles (note that $\mathcal{B}$ does not know the secret key $x = \mathrm{dlog}_g X$ ($y = \mathrm{dlog}_g Y$)). That is, if $\mathrm{DDH}_{g,X}(pk_2, h) = 1$ ($\mathrm{DDH}_{g,Y}(pk_2, h) = 1$), then $h = pk_2^x$ ($h = pk_2^y$) and $\mathcal{B}$ adds $(\mathsf{ID}_1, \mathsf{ID}_2, h, R)$ to the list and returns R. Note that entries of the form $(\mathsf{ID}_1, \mathsf{ID}_2, \perp, R)$ are only added to the list when one of the identities ID_1 or ID_2 is A or B and the other is registered. Finally, if no entry of either of the above forms already appears on the list, that is, no $(\mathsf{ID}_1, \mathsf{ID}_2, h, R)$ and no $(\mathsf{ID}_1, \mathsf{ID}_2, \perp, R)$, then $\mathcal{B}$ picks a random value $R \in \{0,1\}^k$, returns R and adds $(\mathsf{ID}_1, \mathsf{ID}_2, h, R)$ to L.

This completes our description of $\mathcal{B}$'s simulation. Let G denote the event that $\mathcal{A}$ queries the random oracle H with (A, B, h) or (B, A, h) for $h = g^{xy}$, then it efficiently solved $\mathcal{B}$'s own CDH challenge. This can be noticed by $\mathcal{B}$ (with the help of oracle $\mathrm{DDH}_{g,X}(\cdot,\cdot)$ or $\mathrm{DDH}_{g,Y}(\cdot,\cdot)$) and $\mathcal{B}$ can return g^{xy}. Note that if $\mathcal{A}$ does not query H on (A, B, g^{xy}) or (B, A, g^{xy}), then $\mathcal{A}$'s advantage is zero because it cannot disinguish real from random answers to *Test queries.* Hence, if $\mathcal{A}$ has non-negligible success probability, then this query must be made by $\mathcal{A}$. We then see that $\mathrm{Adv}^{\mathrm{dsdh}}_{\mathcal{B},\mathtt{RSAgen}}(k) \geq \mathrm{Adv}^{\text{CKS-light}}_{\mathcal{A},\mathrm{NIKE}_{\mathrm{fac}}} + O(2^{-\delta n(k)})$, where the $O(2^{-\delta n(k)})$ term accounts for the statistical difference between the distribution of g^x and g^y in the real game (where $x, y \in \mathbb{Z}_{\lfloor N/4 \rfloor}$) and the simulation (where $x, y \in \mathbb{Z}_{\phi(N)/4}$). Combining these facts with Theorem 2, we have that $\mathrm{Adv}^{\text{CKS-light}}_{\mathcal{A},\mathrm{NIKE}_{\mathrm{fac}}}(k) \leq \mathrm{Adv}^{\mathrm{fac}}_{\mathcal{C},\mathtt{RSAgen}}(k) + O(2^{-\delta n(k)})$, concluding the proof. □

D Proof of Theorem 4

Proof. Our proof largely follows the IND-CCA security proof for the factoring-based PKE scheme from [26]. Loosely speaking, our system parameters *params* correspond to a PKE public key in the scheme of [26]; NIKE user keys *pk* correspond to PKE ciphertexts; and the NIKE shared key computation roughly corresponds to PKE decryption. The main difference between the NIKE and PKE schemes is that in a NIKE scheme, two user keys are "paired" to generate a shared secret key. In a PKE scheme, a single ciphertext is decrypted "on its own." In particular, there are two NIKE honest identities (for which a Test query is issued), while there is only one PKE challenge ciphertext.

More formally, assume a *CKS-light* adversary $\mathcal{A}$ that only registers consistent keys. We use $\mathcal{A}$ to construct a BBS distinguisher $\mathcal{B}$. Our distinguisher $\mathcal{B}$ gets as input a modulus N, along with a random $D \in \mathbb{QR}_N^+$ and a challenge $C \in \{0,1\}^k$. $\mathcal{B}$'s goal is to distinguish the cases $C = \mathrm{BBS}_N(D^{1/2^k})$ and uniform C. To this end, $\mathcal{B}$ will simulate the *CKS-light* game for $\mathcal{A}$.

First, we will assume that in the original *CKS-light* game, any two public keys (honest or registered by $\mathcal{A}$) lead to the different hash values $t \leftarrow \mathrm{ChamH}_{\mathsf{hk}}(Z||\mathsf{ID}; r)$. A straightforward reduction to the collision resistance of ChamH justifies this assumption and leads to the $\mathrm{Adv}^{\mathrm{coll}}_{\mathcal{A}_{\mathcal{CH}},\mathrm{ChamH}}$ term in (1).

Next, we let $\mathcal{B}$ use the chameleon hash trapdoor ck in its simulation. Concretely, we let $\mathcal{B}$ *initially* uniformly choose the hash values t_1 and t_2 for the two (at that point unknown) honest identitities $\mathsf{ID}_1, \mathsf{ID}_2$. (Later on, when $\mathcal{A}$ decides on ID_1 and ID_2, we let $\mathcal{B}$ use ck to generate ChamH randomness r_1, r_2 such that $t_i = \mathrm{ChamH}_{\mathsf{hk}}(Z_i||\mathsf{ID}_i; r_i)$ for the corresponding public keys.) $\mathcal{B}$ then sets up

$$g = D^{\alpha_1 \cdot \alpha_2} \qquad\qquad u_i = D^{a_i} g^{b_i 2^{3k}} \quad (0 \leq i \leq 2).$$

Here, $\alpha_1, \alpha_2 \leftarrow \mathtt{PrimeGen}(1^{n+k}, \rho)$ (for a uniform $2(n+k)$-bit prime ρ) are primes generated using the assumed algorithm $\mathtt{PrimeGen}$. By assumption about $\mathtt{PrimeGen}$, we have that the $\alpha_i \bmod \rho$ (and thus the $\alpha_i \bmod \phi(N)/4$) are statistically close to uniform. (In particular, we can assume $\alpha_i > 2^{2k}$.) The arising (negligible) statistical defect is accounted for by the $O(2^{-k})$ term in (1). Furthermore, $b_i \in \mathbb{Z}_{\lfloor N/4 \rfloor}$ and $a_i \in \mathbb{Z}_{\lfloor N/4 \rfloor}$ are uniformly chosen, such that $a(t) := a_0 + a_1 t + a_2 t^2 := (t - t_1)(t - t_2) \in \mathbb{Z}[t]$. We will also write $b(t) := b_0 + b_1 t + b_2 t^2 \in \mathbb{Z}[t]$ for brevity. $\mathcal{B}$ invokes $\mathcal{A}$ with the resulting parameters $params = (N, g, u_0, u_1, u_2, \mathsf{hk})$. The honest keys $pk_{\mathsf{ID}_i} = (Z_i, X_i, r_i)$ are computed using

$$Z_i = D^{\alpha_{3-i}} \quad \left(= g^{1/\alpha_i}\right) \qquad\qquad X_i = D^{\alpha_{3-i} \cdot b(t_i)} \quad \left(= (u_0 u_1^{t_i} u_2^{t_i^2})^{\frac{1}{\alpha_i \cdot 2^{3k}}}\right)$$

and randomness r_i that ensures $\mathrm{ChamH}_{\mathsf{hk}}(Z_i||\mathsf{ID}_i; r_i) = t_i$. This implicitly sets $x_i = 1/(\alpha_i \cdot 2^{3k})$, such that the shared key between ID_1 and ID_2 is $\mathrm{BBS}_N(g^{x_1 x_2 2^{5k}}) = \mathrm{BBS}_N(D^{2^{-k}})$. Hence, $\mathcal{B}$'s challenge C can be directly embedded as $\mathcal{A}$'s test challenge key.

It remains to describe how $\mathcal{B}$ answers $\mathcal{A}$'s *Corrupt reveal* queries for $(\mathsf{ID}_i, \mathsf{ID})$ (with $i \in \{1,2\}$ and $\mathsf{ID} \neq \mathsf{ID}_1, \mathsf{ID}_2$). We can assume that ID's public key $pk_{\mathsf{ID}} = (Z, X, r)$ is consistent, so that $Z = g^{x \cdot 2^{3k}}, X = (u_0 u_1^t u_2^{t^2})^x$ for $t = \mathrm{ChamH}_{\mathsf{hk}}(Z||\mathsf{ID}; r)$. To compute the shared key $\mathrm{BBS}_N(g^{(x/\alpha_i)\cdot 2^{2k}}) = \mathrm{BBS}_N(D^{x \cdot \alpha_{3-i} \cdot 2^{2k}})$, it suffices to compute $D^{x \cdot 2^{2k}}$. However, $D^{x \cdot 2^{2k}}$ can be computed from

$$\frac{X}{Z^{b(t)}} = \frac{D^{x \cdot a(t)} g^{x \cdot b(t) \cdot 2^{3k}}}{g^{x \cdot b(t) \cdot 2^{3k}}} = D^{x \cdot a(t)}$$

and $Z = g^{x \cdot 2^{3k}} = D^{x \cdot \alpha_1 \cdot \alpha_2 \cdot 2^{3k}}$, using the extended Euclidean algorithm in the exponent and the fact that $\gcd(a(t), \alpha_1 \cdot \alpha_2 \cdot 2^{3k}) \mid 2^{2k}$ (which holds because $a(t) \leq 2^{2k}$ and the $\alpha_i > 2^{2k}$ are prime).

This completes the description of our simulation. Assuming that no ChamH collision occurs, and ignoring the statistical defect arising from the not quite uniform $\alpha_i \bmod \phi(N)/4$, we get the following:

- when $\mathcal{B}$'s challenge C equals $\mathrm{BBS}_N(D^{2^{-k}})$, then $\mathcal{B}$ simulates the *CKS-light* game with $\mathcal{A}$ and a real challenge key;
- when $\mathcal{B}$'s challenge C is uniform, then $\mathcal{B}$ simulates the *CKS-light* game with $\mathcal{A}$ and a random challenge key.

Thus, $\mathcal{B}$ is a successful BBS distinguisher whenever $\mathcal{A}$ is successful in the *CKS-light* game. (1) follows.

E Proof of Theorem 5

Proof. We proceed via a sequence of games. Let S_i be the event that $\mathcal{A}$ is successful in Game i.

Game 0. Let Game 0 be the original attack game as described in the *CKS-light* security model. By definition, we have that:

$$\mathrm{Adv}^{\text{CKS-light}}_{\mathcal{A},\mathrm{NIKE}_{\text{dbdh-2}}}(k) = |\Pr[S_0] - 1/2|.$$

Game 1. (Eliminate hash collisions.) In this game, the challenger changes its answers to *register corrupt user ID* queries as follows: let A and B be the identities of the two honest users, and let their public keys be (X_A, Z_A, r_A), (X_B, Z_B, r_B) respectively. Let D be the identity of a user that is the subject of a register corrupt user ID query with $pk_D = (X_D, Z_D, r_D)$. If $t_D = \mathrm{ChamH}_{\mathsf{hk}}(Z_D||D; r_D) = \mathrm{ChamH}_{\mathsf{hk}}(Z_A||A; r_A)$ or $t_D = \mathrm{ChamH}_{\mathsf{hk}}(Z_D||D; r_D) = \mathrm{ChamH}_{\mathsf{hk}}(Z_B||B; r_B)$, the challenger aborts (note that in this case the challenger created a collision in $\mathrm{ChamH}_{\mathsf{hk}}$). Otherwise, it continues as in the previous game.

Let $\mathtt{abort}_{\mathtt{ChamH}}$ be the event that a collision was found. Until $\mathtt{abort}_{\mathtt{ChamH}}$ happens, Game 0 and Game 1 are identical. By the *difference lemma* [33], we have

$$|\Pr[S_1] - \Pr[S_0]| \leq \Pr[\mathtt{abort}_{\mathtt{ChamH}}].$$

Furthermore,

$$\Pr[\mathtt{abort}_{\mathtt{ChamH}}] \leq \mathrm{Adv}^{\mathrm{coll}}_{\mathcal{A}_{\mathcal{CH}},\mathrm{ChamH}}(\mathtt{k}).$$

Game 2. In this game a DBDH-2 adversary $\mathcal{B}$ on inputs $(g_2, g_2^a, g_2^b, g_1^c, T) \in \mathbb{G}_2^3 \times \mathbb{G}_1 \times \mathbb{G}_T$, where $a, b, c \in \mathbb{Z}_p$, runs adversary $\mathcal{A}$ against $\mathrm{NIKE}_{\text{dbdh-2}}$ simulating the challenger's behaviour as in Game 1. $\mathcal{B}$'s job is to determine whether T equals $e(g_1, g_2)^{abc}$ or a random element from $\mathbb{G}_T$, where g_2 is a generator of $\mathbb{G}_2$ and $g_1 = \psi(g_2)$ is a generator of $\mathbb{G}_1$.

$\mathcal{B}$ runs $\mathtt{Cham.KeyGen}(1^k)$ to obtain a key pair for a chameleon hash function, $(\mathsf{hk}, \mathsf{ck})$ (here ck is the trapdoor information for the chameleon hash). It then selects $m_1, m_2 \xleftarrow{\$} \{0,1\}^*$ and $r_1, r_2 \xleftarrow{\$} \mathcal{R}_{\mathrm{Cham}}$, where $\mathcal{R}_{\mathrm{Cham}}$ is the chameleon hash function's randomness space. $\mathcal{B}$ computes $t_A = \mathrm{ChamH}_{\mathsf{hk}}(m_1; r_1)$ and $t_B = \mathrm{ChamH}_{\mathsf{hk}}(m_2; r_2)$.

Let $p(t) = p_0 + p_1 t + p_2 t^2$ be a polynomial of degree 2 over $\mathbb{Z}_p$ such that $p(t_A) = p(t_B) = 0$. Let $q(t) = q_0 + q_1 t + q_2 t^2$ be a random polynomial of degree 2 over $\mathbb{Z}_p$. Then $\mathcal{B}$ sets $u_i = (g_1^c)^{p_i} g_1{}^{q_i}$ and

$S = g_1^c$. Since $q_i \xleftarrow{\$} \mathbb{Z}_p$, we have $u_i \xleftarrow{\$} \mathbb{G}_1$. Note that then $u_0 u_1^t u_2^{t^2} = (g_1^c)^{p(t)} {g_1}^{q(t)}$. In particular, $Y_A = g_1^{q(t_A)}$ and $Y_B = g_1^{q(t_B)}$, where $q(t_A)$ and $q(t_B)$ are known values.

$\mathcal{B}$ then answers the following queries:

- *Register honest user ID*: When $\mathcal{B}$ receives a register honest user ID query for identity A from adversary $\mathcal{A}$, it uses the trapdoor information ck of the chameleon hash function to obtain $r_A \in \mathcal{R}_{\text{Cham}}$ such that $\text{ChamH}_{\mathsf{hk}}(g_2^a \| A; r_A) = \text{ChamH}_{\mathsf{hk}}(m_1; r_1) = t_A$. Notice that, according to the definition of chameleon hash functions (see Appendix A), r_A is uniformly distributed over $\mathcal{R}_{\text{Cham}}$ and independent from r_1. Similarly, when $\mathcal{B}$ receives a second register honest user ID query for identity B from $\mathcal{A}$, it obtains $r_B \in \mathcal{R}_{\text{Cham}}$ such that $\text{ChamH}_{\mathsf{hk}}(g_2^b \| B; r_B) = \text{ChamH}_{\mathsf{hk}}(m_2; r_2) = t_B$. Then r_B is also uniformly distributed over $\mathcal{R}_{\text{Cham}}$. Now $\mathcal{B}$ sets:
$$pk_A = ((\psi(g_2^a)^{q(t_A)}, g_2^a, r_A) \text{ and } pk_B = ((\psi(g_2^b)^{q(t_B)}, g_2^b, r_B).$$
These are correct public keys since $p(t_A) = p(t_B) = 0$.
- *Register corrupt user ID*: Here, $\mathcal{B}$ receives a public key pk and a string ID from $\mathcal{A}$, and registers them. As in the original attack game, $\mathcal{B}$ aborts if ID equals one of the honest identities, A or B.
- *Corrupt reveal queries*: In order to output the paired key between one of the two honest users, say A, and a corrupt user, say D, $\mathcal{B}$ first checks if $pk_D = (X_D, Z_D, r_D)$ is a valid public key using the pairing. If not, it rejects the query. This makes sure that pk_D is of the form (Y_D^d, g_2^d, r_D) for some $d \in \mathbb{Z}_p$, where $Y_D = (g_1^c)^{p(t_D)} g_1^{q(t_D)}$ and $r_D \in \mathcal{R}$. This means that $X_D = (g_1^{cd})^{p(t_D)} g_1^{dq(t_D)}$. Thus, g_1^{cd} can be computed from X_D, $Z_D = g_2^d$ and r_D by:
$$g_1^{cd} = (X_D / \psi(Z_D)^{q(t_D)})^{1/p(t_D) \bmod p},$$
where we use the property that $p(t_D) \neq 0 \bmod p$, which follows from the facts that p is a polynomial of degree 2 with roots t_A, t_B and that $t_D \neq t_A, t_B$ (because we have eliminated hash collisions already in Game 1).
Now writing $pk_A = (X_A, Z_A, r_A)$ for the public key of the honest user A, the shared key between A and D can be correctly computed as:
$$K_{A,D} = e(g_1^{cd}, Z_A).$$
- *Test query*: Return T.

This completes our description of $\mathcal{B}$'s simulation. Note that distinguishing the real case from the random case for $\mathcal{A}$ in Game 2 is equivalent to solving the DBDH-2 problem. To see this, note that for user A, we have $Z_A = g_2^a$ and $X_A = \psi(Z_A)^{q(t_A)}$, while for user B, we have $Z_B = g_2^b$ and $X_B = \psi(Z_B)^{q(t_B)}$. Hence $K_{A,B} = e((g_1^c)^b, Z_A) = e((g_1^c)^a, Z_B) = e(g_1, g_2)^{abc}$.

Now, since $\mathcal{B}$'s simulation properly handles all of $\mathcal{A}$'s queries and sets up all values with the correct distributions, we have: $\Pr[S_2] = \Pr[S_1]$.

Game 3. (Replace the challenge.) In this game $\mathcal{B}$ replaces the value T with a random element from $\mathbb{G}_T$. Since T is now completely independent of the challenge bit, we have: $\Pr[S_3] = 1/2$.
Game 2 and Game 3 are identical unless adversary $\mathcal{A}$ can distinguish $e(g_1, g_2)^{abc}$ from a random element. Therefore we have:
$$|\Pr[S_3] - \Pr[S_2]| \leq \text{Adv}_{\mathcal{B},\mathcal{G}2}^{\text{dbdh-2}}(k).$$
By collecting the probabilities relating the different games, we have
$$\begin{aligned}
\text{Adv}_{\mathcal{A},\text{NIKE}_{\text{dbdh-2}}}^{\text{CKS-light}} &= |\Pr[S_0] - 1/2| \\
&\leq |\Pr[S_1] + \text{Adv}_{\mathcal{A}_{\mathcal{CH}},\text{ChamH}}^{\text{coll}}(k) - 1/2| \\
&\leq |\Pr[S_2] + \text{Adv}_{\mathcal{A}_{\mathcal{CH}},\text{ChamH}}^{\text{coll}}(k) - 1/2| \\
&\leq |\Pr[S_3] + \text{Adv}_{\mathcal{B},\mathcal{G}2}^{\text{dbdh-2}}(k) + \text{Adv}_{\mathcal{A}_{\mathcal{CH}},\text{ChamH}}^{\text{coll}}(k) - 1/2| \\
&\leq \text{Adv}_{\mathcal{B},\mathcal{G}2}^{\text{dbdh-2}}(k) + \text{Adv}_{\mathcal{A}_{\mathcal{CH}},\text{ChamH}}^{\text{coll}}(k)
\end{aligned}$$
Thus,
$$\text{Adv}_{\mathcal{B},\mathcal{G}2}^{\text{dbdh-2}}(k) \geq \text{Adv}_{\mathcal{A},\text{NIKE}_{\text{dbdh-2}}}^{\text{CKS-light}}(k) - \text{Adv}_{\mathcal{A}_{\mathcal{CH}},\text{ChamH}}^{\text{coll}}(k).$$
This concludes our proof. $\square$

Remark: We note that the map ψ in $\mathcal{PG}2$ is only used in the security proof for the NIKE scheme $\text{NIKE}_{\text{dbdh-2}}$ and not in the scheme itself.

F Proof of Theorem 6

Proof. Let $\mathcal{A}$ be an adversary against $\mathtt{KEM}(\mathtt{NIKE}, \mathtt{OTS})$. We build $\mathcal{B}$, an adversary against the NIKE scheme in the CKS-light security model.

$\mathcal{B}$, on input *params*, a set of system parameters, picks one identity ID_1 uniformly at random and runs $\mathtt{OTSKeyGen}$ to obtain $(vk, sigk)$. It sets $\mathsf{ID}_2 = vk$ and makes two *register honest user* queries on ID_1 and ID_2 receiving public keys pk_1, pk_2. $\mathcal{B}$ then sets $pk_{\mathtt{KEM}} = (params, \mathsf{ID}_1, pk_1)$. $\mathcal{B}$ also makes a *test* query on $\mathsf{ID}_1, \mathsf{ID}_2$. It receives in reply a value $\hat{K}$, which is either the real key, $K^* = \mathtt{SharedKey}(\mathsf{ID}_1, pk_1, \mathsf{ID}_2, sk_2)$, or a random key K from $\mathcal{SHK}$. $\mathcal{B}$ sets $C^* = (\mathsf{ID}_2, pk_2, \sigma^*)$, where $\sigma^* \leftarrow \mathtt{OTSSign}(sigk, pk_2)$ and gives $(pk_{\mathtt{KEM}}, \hat{K}, C^*)$ to $\mathcal{A}$.

$\mathcal{A}$ now makes $\mathtt{Dec}$ queries which $\mathcal{B}$ handles as follows. For each such query with input C, $\mathcal{B}$ parses C as $(\mathsf{ID}', pk', \sigma')$ and check the signature σ' using $vk' = \mathsf{ID}'$. If $\mathsf{ID}' = \mathsf{ID}_2$ and $(pk', \sigma') \neq (pk_2, \sigma^*)$, then we can build another adversary $\mathcal{C}$ against the strong security of $\mathtt{OTS}$. (This is done using the same simulation as above with the difference that sk_1 and sk_2 are known but vk comes from the OTS experiment. The signing oracle is used to generate σ^* for the challenge ciphertext.) If $\mathsf{ID}' = \mathsf{ID}_1$, it returns $\perp$. Assuming $\mathsf{ID}' \notin \{\mathsf{ID}_1, \mathsf{ID}_2\}$, $\mathcal{B}$ makes a *register corrupt user* query on input ID'. $\mathcal{B}$ then makes a *corrupt reveal* query on ID_1, pk_1, ID', pk' to get either a key $K \in \mathcal{SHK}$, or $\perp$, and returns the result to $\mathcal{A}$.

This completes our description of $\mathcal{B}$'s simulation. $\mathcal{A}$'s view is identical when playing either against $\mathcal{B}$ in this simulation or against its real KEM challenger. Note that in the KEM real game $\mathtt{Dec}(sk_{\mathtt{KEM}}, C)$, where $C = (vk' = \mathsf{ID}', pk', \sigma')$, should return $\mathtt{SharedKey}(\mathsf{ID}', pk', \mathsf{ID}_1, sk_1)$ or $\perp$ if the signature does not verify or if $\mathsf{ID}' = \mathsf{ID}_1$. Note also that in the KEM real game, the challenge query should be answered with either $\mathtt{Enc}(pk_{\mathtt{KEM}}) = (K^*, C^*)$, where $K^* = \mathtt{SharedKey}(\mathsf{ID}_1, pk_1, \mathsf{ID}_2, sk_2)$ and $C^* = (\mathsf{ID}_2, pk_2, \sigma^*)$, or a pair (K, C^*), with K chosen at random from $\mathcal{SHK}$. This is exactly what is done in $\mathcal{B}$'s simulation.

Whenever $\mathcal{A}$ outputs a bit $\hat{b}$, $\mathcal{B}$ outputs the same bit. Then we have that $\mathcal{B}$'s advantage in breaking the NIKE scheme is the same as $\mathcal{A}$'s in breaking the KEM. Counting queries made $\mathcal{B}$ in response to $\mathcal{A}$'s queries completes the proof. $\square$