Universally Composable Non-Interactive Key Exchange*

Eduarda S.V. Freire[†1], Julia Hesse[2], and Dennis Hofheinz[‡2]

[1]Royal Holloway, University of London, United Kingdom
Eduarda.Freire.2009@live.rhul.ac.uk
[2]Karlsruhe Institute of Technology, Germany
{julia.hesse,dennis.hofheinz}@kit.edu

Abstract

We consider the notion of a *non-interactive key exchange (NIKE)*. A NIKE scheme allows a party A to compute a common shared key with another party B from B's public key and A's secret key alone. This computation requires no interaction between A and B, a feature which distinguishes NIKE from regular (i.e., interactive) key exchange not only quantitatively, but also qualitatively.

Our first contribution is a formalization of NIKE protocols as ideal functionalities in the Universal Composability (UC) framework. As we will argue, existing NIKE definitions (all of which are game-based) do not support a modular analysis either of NIKE schemes themselves, or of the use of NIKE schemes. We provide a simple and natural UC-based NIKE definition that allows for a modular analysis both of NIKE schemes and their use in larger protocols.

We proceed to investigate the properties of our new definition, and in particular its relation to existing game-based NIKE definitions. We find that

(a) game-based NIKE security is equivalent to UC-based NIKE security against *static* corruptions, and
(b) UC-NIKE security against adaptive corruptions *cannot* be achieved without additional assumptions (but *can* be achieved in the random oracle model).

Our results suggest that our UC-based NIKE definition is a useful and simple abstraction of non-interactive key exchange.

Keywords: non-interactive key exchange, universal composability.

1 Introduction

Non-interactive key exchange. In a non-interactive key exchange (NIKE) scheme, any two parties can compute a common shared key without any interaction. Concretely, a NIKE scheme enables a party A to compute a shared key $K_{A,B} = K_{B,A}$ with party B from A's secret key sk_A and B's public key pk_B. A very simple (albeit only mildly secure) example of a NIKE scheme is the Diffie-Hellman key exchange protocol [14]. (Here, $K_{A,B} = g^{ab}$ can be computed from $\mathsf{sk}_A = a$ and $\mathsf{pk}_B = g^b$, or from $\mathsf{pk}_A = g^a$ and $\mathsf{sk}_B = b$.)

A NIKE scheme offers guarantees that are quite different from a regular (i.e., interactive) key exchange (KE) protocol: a NIKE scheme can only offer one session (i.e., shared key) per pair of

*An extended abstract of this work will appear in the proceedings of SCN 2014. This is the full version.

[†]Supported by CAPES Foundation/Brazil on grant 0560/09-0 and Royal Holloway, University of London
[‡]Supported in part by DFG grant GZ HO 4534/4-1.

public keys. On the other hand, in NIKE schemes, the notion of (specifically adversarial) key registrations plays a crucial role. Namely, while KE schemes use long-term public keys commonly only to achieve authentication properties (e.g., [7, 8]), a NIKE scheme must completely rely on public and secret keys. In return, a NIKE scheme offers its functionality without any interaction – a feature that has found use, e.g., in constructions of PKE schemes [18], designated verifier signature schemes [25], deniable authentication [15], or in wireless and sensor networks [20]. To see how the latter could benefit from the non-interactivity we refer to [11], where it is shown that the energy costs of communication can be significantly reduced when using non-interactive key exchange schemes rather than interactive ones. Moreover, as a further application, NIKE can even be used as a basis for interactive key exchange [2]. We stress that in many of the above mentioned applications [18, 25, 15], non-interactivity is a crucial requirement, and not only an efficiency bonus. We believe that this justifies the investigation of NIKE schemes as such.

Previous NIKE definitions. Somewhat surprisingly, the syntax and security of NIKE schemes has been formalized only very recently, by Cash, Kiltz, and Shoup [12].[1] They also construct an efficient NIKE scheme in the random oracle model, based on the Computational Diffie-Hellman assumption. Further constructions and variants of the NIKE definition were given by Freire et al. [18]. All of the security definitions in [12, 18] are game-based and do not consider the registration process of public keys itself. This is a bit unfortunate, in particular since a factoring-based NIKE scheme from [18] explicitly requires a nontrivial key registration (and can thus not be completely modeled in the setting of [18]).

In fact, game-based security definitions (like those from [12, 18]) appear unsuitable to model an interactive key registration process for NIKE schemes. Namely, adding a (presumably adversarially controlled) interactive message scheduling would considerably complicate the clean and simple NIKE definitions of [12, 18]. Indeed, it seems more natural and modular to conceptually separate the interactive key registration process from the actual security of (non-interactive) NIKE sessions. However, it is not obvious how to achieve such a conceptual separation using a game-based security definition.

Our contribution. In this work, we devise a simple and intuitive NIKE definition that enables a modular analysis both of NIKE schemes themselves and their use in larger protocols. Our definition is set in the framework of Universal Composability (UC) [4],[2] which allows for a convenient separation of the interactive key registration phase and the actual NIKE scheme. Specifically, we can analyze key registration and NIKE scheme (assuming correctly registered public keys) separately. Besides, a formalization as an ideal functionality in the UC framework yields a very natural and intuitive characterization of a NIKE scheme.

We demonstrate the usefulness of our definition by showing that our definition can be seen as a generalization of existing game-based NIKE definitions, and that the factoring-based NIKE scheme of [18] can be analyzed with respect to our NIKE definition. This in particular means that, while being conceptually simpler, our NIKE notion retains a form of backward compatibility with existing notions.

Why KE functionalities are not suitable for NIKE protocols. Existing KE functionalities (e.g., [8, 24]) are designed for interactive key exchange protocols and allow multiple sessions per pair

[1] A formalization of NIKE schemes as variants of interactive KE schemes (e.g., using the KE security models of [3, 30, 7, 8]) seems possible; however, as argued above, a case-tailored NIKE definition would appear simpler and more useful.

[2] More specifically, we use the variant "GNUC" [23] of UC.

of public keys. In contrast, our own NIKE functionality is a non-interactive (i.e., immediate) and supports only one session per pair of public keys. As mentioned above, non-interactivity is crucial in certain applications (e.g., [18, 25, 20, 15]); however, currently no NIKE schemes that support multiple sessions per pair of public keys are known. (Hence we have restricted our functionality to one session per pair of public keys.)

Of course, one could modify existing KE functionalities by restricting their usage to one session, or by modifying the adversary to immediately deliver outputs. This would have essentially the same effect as our tailor-made NIKE functionality. We believe, however, that a specific NIKE functionality is simpler, and the conceptual differences between (interactive) KE and NIKE justify a separate functionality.

Some technical details. As already explained, we formalize a NIKE scheme itself and the key registration process separately. Concretely, we consider three ideal functionalities (in the sense of UC), $\mathcal{F}_{CRS}$, $\mathcal{F}_{NIKE}$ and $\mathcal{F}_{KR}$. $\mathcal{F}_{CRS}$ provides a common reference string (CRS), which abstracts the availability of public parameters for the NIKE scheme. $\mathcal{F}_{KR}$ abstracts the key registration process. We stress that, similar to [26], and unlike [1, 10], our $\mathcal{F}_{KR}$ functionality allows a party to register arbitrary public keys (that pass some – possibly interactive – validity check). In particular, $\mathcal{F}_{KR}$ does not choose key pairs for a party. This yields a weaker, but arguably more realistic abstraction of key registration. Indeed, we will not attempt to implement $\mathcal{F}_{KR}$ itself – rather, we view $\mathcal{F}_{KR}$ as an abstraction of an actual key registration authority.

$\mathcal{F}_{NIKE}$, on the other hand, completely abstracts a NIKE scheme. Hence, a NIKE scheme may (or may not) implement $\mathcal{F}_{NIKE}$ in the $(\mathcal{F}_{CRS}, \mathcal{F}_{KR})$-hybrid model (i.e., using an instance of $\mathcal{F}_{CRS}$ and $\mathcal{F}_{KR}$). In a nutshell, $\mathcal{F}_{NIKE}$ simply provides every pair of parties with a single, independently uniform shared key K.

We first show that our formalization can be seen as a generalization of the previous game-based definitions of [12, 18]. Concretely, let us call a NIKE scheme `NIKE` *CKS-secure* if it achieves the game-based notion of [12]. We show (in Appendix B) that the *CKS* notion and the NIKE security notions from [18] are all polynomially equivalent even if we allow some special types of re-registration of users that are not allowed in those models. Since [12, 18] do not model key registration, we must assume that `NIKE` runs with a trivial key registration in which parties simply send their public keys to $\mathcal{F}_{KR}$, and no validity check whatsoever is performed. We show that

(a) `NIKE` is *CKS*-secure if and only if `NIKE` securely realizes $\mathcal{F}_{NIKE}$ (with respect to the trivial key registration described above) against *static*[3] adversaries,

(b) $\mathcal{F}_{NIKE}$ cannot be realized without additional (e.g., set-up) assumptions against *adaptive* adversaries, but

(c) if `NIKE` is *CKS*-secure,[4] then a variant of `NIKE` with hashed shared keys securely realizes $\mathcal{F}_{NIKE}$ against adaptive adversaries in the random oracle model.

Results (b) and (c) resemble similar results by Nielsen [27] for the case of UC-secure public-key encryption. Specifically, to show (b), we show that a UC simulator $\mathcal{S}$ (as necessary to show UC security) attempting to emulate an adaptive attack on `NIKE` may run into a commitment problem. (We note, however, that the commitment problem we encounter for NIKE schemes is slightly different from the one for public-key encryption from [27]; see Section 4.2.1 for details.)

Secondly, we remark that $\mathcal{F}_{NIKE}$ and $\mathcal{F}_{KR}$ allow to model NIKE schemes that cannot be modeled using previous NIKE notions. Specifically, we observe that the key registration of the factoring-

[3] However, our result hinges on the exact definition of "static corruptions" — see Section 4.1 for details.

[4] Actually we only need a weaker version of this notion, which is "search-based" instead of indistinguishability-based.

based NIKE scheme from [18] can be handled using $\mathcal{F}_{KR}$.

Further related work. Apart from the mentioned works dealing with *public-key-based* NIKE schemes, the concept of NIKE has also been considered in the *identity-based* setting (e.g., [29, 16, 28, 20, 19]).

Roadmap. After recalling (and slightly adapting) previous NIKE definitions in Section 2, we present our UC-based NIKE definition in Section 3. We investigate the properties of our new definition in Section 4. Namely, Section 4.1, relates our definition (restricted to static corruptions) to existing (game-based) definitions. Section 4.2 and Section 4.4 contain our results for adaptive corruptions: Section 4.2 shows that adaptive UC-based NIKE security cannot be achieved without additional (e.g., setup) assumptions, and Section 4.4 describes a simple transformation that achieves adaptive UC-based NIKE security in the random oracle model. Due to lack of space, we postpone a complete modelling of the factoring-based NIKE scheme from [18] to Appendix A. We deliver more details about the game-based NIKE notions (and our adaptations) in Appendix B. The remaining appendix recalls some details about the UC model (Appendices C,D, and E), and delivers full versions of proofs that are only sketched in the main part (Appendices F and G).

2 Preliminaries

NIKE schemes. Following [12], and later [18], we formally define non-interactive key exchange in the *public key setting*. A non-interactive key exchange scheme `NIKE` in the public key setting consists of three algorithms: `NIKE.CommonSetup`, `NIKE.KeyGen` and `NIKE.SharedKey`. The first algorithm is run by a trusted authority, while the second and third algorithms can be run by any user.

- $\texttt{NIKE.CommonSetup}(1^k)$: This algorithm is probabilistic and takes as input a security parameter k. It outputs a set of system parameters, *params*.
- $\texttt{NIKE.KeyGen}(\mathit{params}, \mathsf{ID})$: This is the key generation algorithm, a probabilistic algorithm that on inputs *params* and a user identifier $\mathsf{ID} \in \mathcal{IDS}$, where $\mathcal{IDS}$ is an identity space, outputs a public key/secret key pair $(\mathsf{pk}, \mathsf{sk})$.
- $\texttt{NIKE.SharedKey}(\mathsf{ID}_1, \mathsf{pk}_1, \mathsf{ID}_2, \mathsf{sk}_2)$: On inputs a user identifier $\mathsf{ID}_1 \in \mathcal{IDS}$ and a public key pk_1 along with another user identifier $\mathsf{ID}_2 \in \mathcal{IDS}$ and a secret key sk_2, this deterministic algorithm outputs a shared key in $\mathcal{SHK}$, the shared key space, for the two users, or a failure symbol $\perp$. We assume that this algorithm outputs $\perp$ if $\mathsf{ID}_1 = \mathsf{ID}_2$ or if any of its input is missing or is not in the correct domain.

For correctness, for any pair of user identifiers $\mathsf{ID}_1, \mathsf{ID}_2$, and corresponding public key/secret key pairs $(\mathsf{pk}_1, \mathsf{sk}_1)$, $(\mathsf{pk}_2, \mathsf{sk}_2)$, `NIKE.SharedKey` satisfies

$$\texttt{NIKE.SharedKey}(\mathsf{ID}_1, \mathsf{pk}_1, \mathsf{ID}_2, \mathsf{sk}_2) = \texttt{NIKE.SharedKey}(\mathsf{ID}_2, \mathsf{pk}_2, \mathsf{ID}_1, \mathsf{sk}_1).$$

Throughout the paper we will be considering, w.l.o.g., a shared key space $\mathcal{SHK} = \{0,1\}^k$.

(Game-based) security of NIKE. Several game-based security notions for NIKE, where an adversary against a NIKE scheme is required to distinguish real from random keys, were presented in [18]. The security notions in [18] are denoted by *CKS-light*, *CKS*, *CKS-heavy* and *m-CKS-heavy*. In those notions, minimal assumptions are made about the Certificate Authority (CA) in the PKI supporting the non-interactive key exchange. The security models in [18] do not rely on the CA checking 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 secret key. An adversary against a NIKE scheme in those models is thus allowed to introduce arbitrary public keys (for which it might not know the corresponding secret keys) into the system. However, the security models in [18] do not capture re-registration of (honest or corrupted) users, (i.e., when a user renews its public key).

In this paper, we make use of some of the security notions from [18], but in the more realistic scenario where users are allowed to re-register public keys with a CA, and an adversary against a NIKE scheme is allowed to re-register a user as corrupted even if it was registered as honest before. Also, for some cases we need weaker security notions, where an adversary instead of being required to distinguish real from random keys, it is required to actually output the shared key between two honest users.

We present in Appendix B the security models from [18] as well as our augmented versions of those models, allowing honest re-registrations of users and corrupt registration of previously registered honest users. Moreover, in Appendix B, we prove the equivalence of all the above mentioned security models *with or without* the re-registration of users just described. Additionally, a weaker, search-based (instead of indistinguishability-based) security notion is also described there. Throughout the paper we add $^{+}$ to the notation of the security models from [18] to denote the augmented versions of those models when re-registration of honest users is allowed; We add $^{++}$ to the notation to denote that both re-registration of honest users, as well as corrupt registration of previously registered honest users, are allowed.

3 NIKE in the UC model

NIKE in the UC model of protocol execution. In order to establish relationships between game-based NIKE security notions and UC notions, we first explain how parties behave in a real execution of a NIKE scheme $\texttt{NIKE}$ with environment $\mathcal{Z}$ and adversary $\mathcal{A}$, in the hybrid-model with a key registration functionality $\mathcal{F}_{KR}^{f}$ (described below) and a common reference string functionality $\mathcal{F}_{CRS}$ (described in Appendix E). For a more detailed description of how the UC model of protocol execution works, we refer the reader to Appendix C.

A party P_i proceeds as follows, running with $\mathcal{Z}$, $\mathcal{A}$, $\mathcal{F}_{KR}^{f}$ and $\mathcal{F}_{CRS}$:

- Upon receipt of $(\texttt{register}, \mathsf{P}_i)$ from $\mathcal{Z}$ for the first time, request *params* from the $\mathcal{F}_{CRS}$ functionality and run $\texttt{NIKE.KeyGen}(\mathit{params}, \mathsf{P}_i)$ to generate a public key/secret key pair $(\mathsf{pk}_i, \mathsf{sk}_i)$. Register the public key pk_i with $\mathcal{F}_{KR}^{f}$ by sending $(\texttt{register}, \mathsf{P}_i, \mathsf{pk}_i, \tau)$, where τ is a proof of validity of pk_i[5].
- On input $(\texttt{init}, \mathsf{P}_i, \mathsf{P}_j)$ from $\mathcal{Z}$, request the public key corresponding to party P_j by sending $(\texttt{lookup}, \mathsf{P}_j)$ to $\mathcal{F}_{KR}^{f}$. Compute $K_{i,j} \leftarrow \texttt{NIKE.SharedKey}(\mathsf{P}_j, \mathsf{pk}_j, \mathsf{P}_i, \mathsf{sk}_i)$ and send $(\mathsf{P}_i, \mathsf{P}_j, K_{i,j})$ to $\mathcal{Z}$.
- On input $(\texttt{renew}, \mathsf{P}_i)$ from $\mathcal{Z}$ compute $(\mathsf{pk}_i, \mathsf{sk}_i) \xleftarrow{\$} \texttt{NIKE.KeyGen}(\mathit{params}, \mathsf{P}_i)$ to generate a new public key/secret key pair. Register pk_i with $\mathcal{F}_{KR}^{f}$ as before.
- Upon receipt of $(\texttt{corrupt}, \mathsf{P}_i)$ from $\mathcal{A}$, send the entire current state to $\mathcal{A}$ and from this point on relay everything to or from $\mathcal{A}$.

Ideal functionalities for NIKE. We now introduce our ideal functionalities $\mathcal{F}_{NIKE}$ and $\mathcal{F}_{KR}^{f}$. $\mathcal{F}_{NIKE}$ abstracts the task of non-interactive key exchange. To enable a modular analysis of NIKE

[5] How such a proof τ looks like depends on the concrete NIKE scheme. For instance, in most existing NIKE protocols, the proof will be trivial (i.e., empty), since the validity of public keys is publicly verifiable.

schemes we separate the process of key registration from the issuance of shared keys and, in addition to $\mathcal{F}_{NIKE}$, introduce a separate ideal functionality $\mathcal{F}_{KR}^{f}$ for the task of public key registration.

The key exchange functionality $\mathcal{F}_{NIKE}$. Our ideal functionality $\mathcal{F}_{NIKE}$ is suitable for non-interactive key exchange in the public key setting. $\mathcal{F}_{NIKE}$ handles the generation of shared keys between two parties, providing the security guararantees of non-interactive key exchange: if an honest party P_i obtained a key $K_{i,j}$ from a session with an honest party P_j, then $K_{i,j}$ is ideally random and unknown to the adversary. Also, $\mathcal{F}_{NIKE}$ requires the requesting party to know the identity of the peer. We stress that $\mathcal{F}_{NIKE}$ can also handle issuance of new shared keys (e.g., after a party renews its public key in a real NIKE scheme). If one of the parties is corrupted by the time that a request was made, then there is no guarantee of security of the shared key.

We remark that, as standard in the GNUC model, $\mathcal{F}_{NIKE}$'s output towards the parties is scheduled *immediately*, i.e., without adversarial intervention. Even more, we model $\mathcal{F}_{NIKE}$ such that the computation of shared keys between honest parties is completely oblivious to the adversary. This models the fact that a party, when using a non-interactive key exchange scheme, needs to be able to perform this computation without the help of other parties. Modeling $\mathcal{F}_{NIKE}$ as immediate enforces this, because the execution of an interactive protocol can be delayed by the real-world adversary and is thus not simulatable in the ideal world, where the adversary has no ability to schedule $\mathcal{F}_{NIKE}$'s output. We note that even in this setting, during computation of a shared key, a real-world party is still able to use hybrid ideal functionalities that do not communicate with the adversary.

$\mathcal{F}_{NIKE}$ operates in two modes, depending on the kind of session for which it should output a key. We call a session honest if both parties are honest, otherwise the session is called corrupted. We assume $\mathcal{F}_{NIKE}$ knows which parties are corrupted.[6] Additionally, $\mathcal{F}_{NIKE}$ maintains three lists:

- a list Λ_{renew} to store parties that want to renew their public key/secret key pair;
- a list Λ_{reg} to store parties that successfully registered a public key;
- a list Λ_{keys} to store shared keys for pairs of parties.

We note that technically, sessions with dishonest parties who still maintain an honestly registered key could alternatively be treated as honest. This yields an alternative ideal functionality that provides slightly better security guarantees than $\mathcal{F}_{NIKE}$ at the cost of a more complicated description. (Our proofs below carry over to such an alternative functionality.)

On the immediateness of $\mathcal{F}_{NIKE}$. As specified above, $\mathcal{F}_{NIKE}$ does *not* guarantee immediate output upon an `init` query that refers to a corrupted session. Namely, in that case, the adversary is queried for a key $K_{i,j}$, and could potentially block $\mathcal{F}_{NIKE}$'s output by not sending that key. (We stress that the simulators we construct will never block immediate delivery of keys in this sense.) To avoid this possibility to block outputs, we could have let the adversary upload an algorithm AdvKey to $\mathcal{F}_{NIKE}$ that is used to immediately derive keys $K_{i,j} := \mathsf{AdvKey}(\mathsf{P}_i, \mathsf{P}_j)$ without querying the adversary. (This is in analogy to similar algorithms in signature and encryption functionalities [5, 6].) While possible, this would entail technical complications (such as communicating code and an AdvKey function that will have to use a pseudorandom function to derive keys), so we keep the slightly simpler and more intuitive formulation from above.

[6] This assumption is standard in UC (e.g., [9, 8]) and implemented as part of the model of computation. However, since the corruption mechanism is not fully specified in GNUC (yet), we simply assume a mechanism. (For concreteness, we assume that ideal functionalities send any party a special "`corrupted?`" request that is automatically and directly answered with "yes" if and only if that party has been corrupted.)

$\mathcal{F}_{NIKE}$ proceeds as follows, running on security parameter k, with parties $\mathsf{P}_1, \ldots, \mathsf{P}_n$ and an adversary.

- On input $(\texttt{register}, \mathsf{P}_i)$ from P_i forward $(\texttt{register}, \mathsf{P}_i)$ to the adversary.
- On input $(\mathsf{P}_i, \texttt{registered})$ from the adversary, if $\mathsf{P}_i \notin \Lambda_{reg}$, add P_i to Λ_{reg}. Else, if $\mathsf{P}_i \in \Lambda_{renew}$, delete every existing entry $(\{\mathsf{P}_i, \cdot\}, key)$ from Λ_{keys} and delete P_i from Λ_{renew}. In any case, send $(\mathsf{P}_i, \texttt{registered})$ back to the adversary.
- On input $(\texttt{init}, \mathsf{P}_i, \mathsf{P}_j)$ from P_i, if $\mathsf{P}_j \notin \Lambda_{reg}$, return $(\mathsf{P}_i, \mathsf{P}_j, \perp)$ to P_i. If $\mathsf{P}_j \in \Lambda_{reg}$, we consider two cases:
  - Corrupted session mode: if there exists an entry $(\{\mathsf{P}_i, \mathsf{P}_j\}, K_{i,j})$ in Λ_{keys}, set $key = K_{i,j}$. Else send $(\texttt{init}, \mathsf{P}_i, \mathsf{P}_j)$ to the adversary. After receiving $(\{\mathsf{P}_i, \mathsf{P}_j\}, K_{i,j})$ from the adversary, set $key = K_{i,j}$ and add $(\{\mathsf{P}_i, \mathsf{P}_j\}, key)$ to Λ_{keys}.
  - Honest session mode: if there exists an entry $(\{\mathsf{P}_i, \mathsf{P}_j\}, K_{i,j})$ in Λ_{keys}, set $key = K_{i,j}$, else choose $key \xleftarrow{\$} \{0,1\}^k$ and add $(\{\mathsf{P}_i, \mathsf{P}_j\}, key)$ to Λ_{keys}.

  Return $(\mathsf{P}_i, \mathsf{P}_j, key)$ to P_i.
- On input $(\texttt{renew}, \mathsf{P}_i)$ from P_i, store P_i in Λ_{renew} and forward $(\texttt{renew}, \mathsf{P}_i)$ to the adversary.

Description of the ideal functionality $\mathcal{F}_{NIKE}$.

The key registration functionality $\mathcal{F}_{KR}^f$. The ideal functionality for key registration is motivated by the key registration process in the real world, which is usually operated by a trusted authority, e.g., a CA. We can assume authenticated channels between each party and the CA (because usually a CA requires a proof of identity, e.g. possession of an identity card or, for remote use, a valid signature). Using standard techniques (e.g., public-key encryption), we can then establish secure channels between party and CA. Note that, even with secure channels, the adversary still learns about registrations taking place and is able to delay them. This leads to the following ideal functionality:

$\mathcal{F}_{KR}^f$ proceeds as follows, running with parties $\mathsf{P}_1, \ldots, \mathsf{P}_n$ and an adversary.

- On input $(\texttt{register}, \mathsf{P}_i, \mathsf{pk}_i, \tau)$ from P_i send $(\texttt{register}, \mathsf{P}_i)$ to the adversary.
- On input $(\texttt{output}, \mathsf{P}_i)$ from the adversary, if $f(\mathsf{P}_i, \mathsf{pk}_i, \tau) = 0$, send $\perp$ to P_i. Otherwise, store $(\mathsf{P}_i, \mathsf{pk}_i)$ and send $(\mathsf{P}_i, \mathsf{pk}_i, \texttt{registered})$ to P_i.
- On input $(\texttt{lookup}, \mathsf{P}_i)$ return $(\mathsf{P}_i, \mathsf{pk}_i)$. If this entry does not exist, return $\perp$.

Description of the ideal functionality $\mathcal{F}_{KR}^f$.

$\mathcal{F}_{KR}^f$ is provided with an efficiently computable function f that takes as input a party identifier P_i, a public key pk_i and a proof of validity, τ, of the public key. f returns 1 if τ is a valid proof for pk_i, and 0 otherwise. The adversary obtains a notification from $\mathcal{F}_{KR}^f$ when a party tries to register and needs to send a notification back so that $\mathcal{F}_{KR}^f$ can proceed. This models the fact that the output of the functionality can be delayed by the adversary.

Note that the function f needs to be specified and can be used to obtain different ideal functionalities. For example, if we want $\mathcal{F}_{KR}^f$ to accept all public keys, we can set f to be constant,

e.g. $f \equiv 1$. We denote this special functionality by $\mathcal{F}_{KR}^{1}$ and allow omitting τ in the inputs for $\mathcal{F}_{KR}^{1}$. We explicitly allow interactive key registrations (i.e., implementations of $\mathcal{F}_{KR}^{f}$) – only the ideal functionality $\mathcal{F}_{KR}^{f}$ uses f to (non-interactively) check validity of keys. (Hence, an interactive key registration protocol could enable a simulator to extract a witness for f.)

Finally, we remark that we explicitly do not require proofs of possession (of secret keys), as popular in concrete public-key infrastructures. However, proofs of possession can be seen as a special case of $\mathcal{F}_{KR}^{f}$ (in which τ simply is the secret key for pk, which can be verified by a suitable f).

4 Results

4.1 Static corruption

We show that any CKS^{+}-secure NIKE scheme $\mathtt{NIKE}$ emulates the functionality $\mathcal{F}_{NIKE}$ in a hybrid UC model, if and only if the environment $\mathcal{Z}$ is restricted to static corruptions. (With static corruptions, we mean that a party can only be corrupted before it obtains any protocol input from $\mathcal{Z}$. However, we point out that there is a subtlety regarding the precise definition of static corruptions – see the comment after the proof of Theorem 4.2.)

We remind the reader that the CKS^{+} security notion is an augmented version of the CKS security notion from [18] including honest re-registration of parties (see Section 2 and Appendix B for definitions).

Theorem 4.1. *Let $\mathtt{NIKE}$ be a CKS^{+}-secure NIKE scheme. Then $\mathtt{NIKE}$ realizes $\mathcal{F}_{NIKE}$ in the $(\mathcal{F}_{CRS},\mathcal{F}_{KR}^{1})$-hybrid model with respect to static corruptions.*

Proof. It suffices to show that there exists a simulator $\mathcal{S}$ for the dummy adversary $\mathcal{A}$. $\mathcal{S}$ interacts with an environment $\mathcal{Z}$ and $\mathcal{F}_{NIKE}$. $\mathcal{S}$ maintains a list of corrupted parties and a list Λ with entries of the form $(\mathsf{P}_i, \mathsf{pk}_i, \mathsf{sk}_i)$, containing party identifiers and their public key/secret key pairs. For every party only the newest entry is kept. Thus, there is at most one entry for each party identifier P_i. A party's entry contains a public key if and only if it successfully registered this key with $\mathcal{F}_{KR}^{1}$. We specify the reactions of $\mathcal{S}$ to invocations from $\mathcal{Z}$ and $\mathcal{F}_{NIKE}$:

($\mathtt{parameters}$) **from $\mathcal{Z}$.** $\mathcal{Z}$ issues this request to the adversary because it cannot access $\mathcal{F}_{CRS}$ directly. $\mathcal{S}$ simulates $\mathcal{F}_{CRS}$ by obtaining $params \xleftarrow{\$} \mathtt{NIKE.CommonSetup}(1^k)$ once and, from then on, $\mathcal{S}$ always answers this request with $params$.

($\mathtt{register}, \mathsf{P}_i$) **from $\mathcal{F}_{NIKE}$.** We may assume that $\mathcal{S}$ already computed $params$. If Λ contains no entry $(\mathsf{P}_i, \cdot, \cdot)$, $\mathcal{S}$ obtains $(\mathsf{pk}_i, \mathsf{sk}_i) \xleftarrow{\$} \mathtt{NIKE.KeyGen}(params, \mathsf{P}_i)$ and stores $(\mathsf{P}_i, \mathsf{pk}_i, \mathsf{sk}_i)$ in Λ. $\mathcal{S}$ then sends a message $(\mathsf{P}_i, \mathtt{registered})$ to $\mathcal{F}_{NIKE}$, waits for $(\mathsf{P}_i, \mathtt{registered})$ from $\mathcal{F}_{NIKE}$ and sends $(\mathtt{register}, \mathsf{P}_i)$ to $\mathcal{Z}$ (simulating that message from $\mathcal{F}_{KR}^{1}$ to the dummy adversary $\mathcal{A}$).

($\mathtt{init}, \mathsf{P}_i, \mathsf{P}_j$) **from $\mathcal{F}_{NIKE}$.** The receipt of this input implies that $\mathsf{P}_i, \mathsf{P}_j$ are not both honest. We may assume P_j is corrupted, because $\mathcal{S}$ would not send $(\mathtt{init}, \mathsf{P}_i, \mathsf{P}_j)$ through a corrupted P_i. Finally, $\mathcal{S}$ returns $(\mathsf{P}_i, \mathsf{P}_j, \mathtt{NIKE.SharedKey}(\mathsf{P}_j, \mathsf{pk}_j, \mathsf{P}_i, \mathsf{sk}_i))$ to $\mathcal{F}_{NIKE}$ (note that this output could be $\perp$).

($\mathtt{renew}, \mathsf{P}_i$) **from $\mathcal{F}_{NIKE}$.** $\mathcal{S}$ obtains $(\mathsf{pk}_i, \mathsf{sk}_i) \xleftarrow{\$} \mathtt{NIKE.KeyGen}(params, \mathsf{P}_i)$ and stores $(\mathsf{P}_i, \mathsf{pk}_i, \mathsf{sk}_i)$ in Λ, overwriting any existing entry for P_i if necessary. $\mathcal{S}$ then sends $(\mathsf{P}_i, \mathtt{registered})$ to $\mathcal{F}_{NIKE}$, waits for $(\mathsf{P}_i, \mathtt{registered})$ from $\mathcal{F}_{NIKE}$ and sends $(\mathtt{register}, \mathsf{P}_i)$ to $\mathcal{Z}$.

($\mathtt{corrupt}, \mathsf{P}_i$) **from $\mathcal{Z}$.** Again, $\mathcal{Z}$ will issue this request to the adversary. $\mathcal{S}$ corrupts P_i and adds P_i to its list of corrupted parties.

$(\texttt{register}, \mathsf{P}_i, \mathsf{pk}_i)$ **from** $\mathcal{Z}$**.** Such a request will only be made by $\mathcal{Z}$ to the adversary, which is asked to let a corrupted party P_i register pk_i as its public key. $\mathcal{S}$ then stores $(\mathsf{P}_i, \mathsf{pk}_i, \perp)$ in Λ, sends $(\mathsf{P}_i, \texttt{registered})$ to $\mathcal{F}_{NIKE}$, waits for $(\mathsf{P}_i, \texttt{registered})$ from $\mathcal{F}_{NIKE}$ and sends $(\texttt{register}, \mathsf{P}_i)$ to $\mathcal{Z}$. $\mathcal{S}$ then sends $(\texttt{register}, \mathsf{P}_i)$ to $\mathcal{Z}$, $(\mathsf{P}_i, \texttt{registered})$ to $\mathcal{F}_{NIKE}$ and stores $(\mathsf{P}_i, \mathsf{pk}_i, \perp)$ in Λ.

$(\texttt{lookup}, \mathsf{P}_i)$ **from any entity.** If Λ contains an entry $(\mathsf{P}_i, \mathsf{pk}_i, \cdot)$ return $(\mathsf{P}_i, \mathsf{pk}_i)$, else return $\perp$.

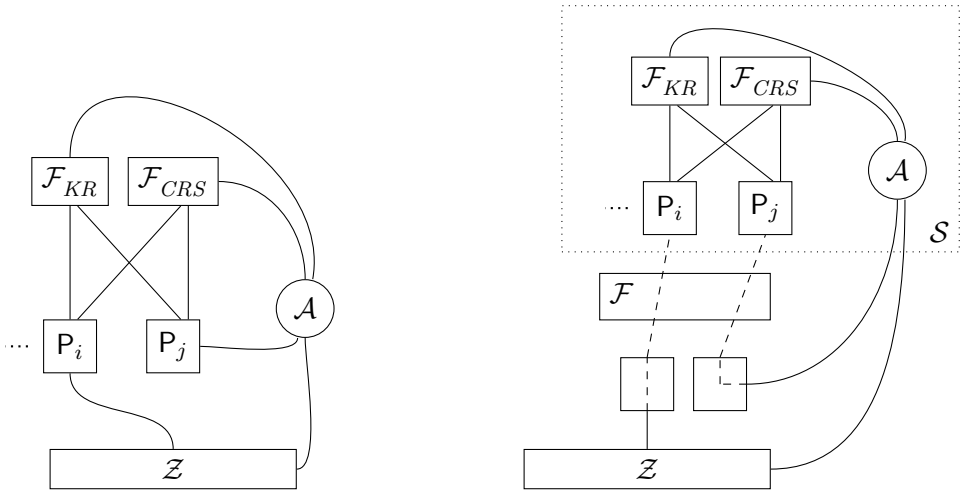

Figure 1: Transition from G_0 (left) to G_1 (right)

Now let $\mathcal{A}$ be the dummy adversary and $\texttt{NIKE}$ a CKS^+-secure NIKE scheme. We show that for every environment $\mathcal{Z}$

$$\text{Exec}[\mathcal{F}_{NIKE}, \mathcal{S}, \mathcal{Z}] \approx \text{Exec}[\texttt{NIKE}, \mathcal{A}, \mathcal{Z}].$$

Here $\text{Exec}[\texttt{NIKE}, \mathcal{A}, \mathcal{Z}]$ (resp. $\text{Exec}[\mathcal{F}_{NIKE}, \mathcal{S}, \mathcal{Z}]$) denotes the random variable describing the output of environment $\mathcal{Z}$ when interacting with adversary $\mathcal{A}$ (resp. $\mathcal{S}$) and protocol $\texttt{NIKE}$ (resp. functionality $\mathcal{F}_{NIKE}$).[7]

Game G_0: Real protocol run. This is the real execution of $\texttt{NIKE}$ with dummy adversary $\mathcal{A}$. A specific instance of this game is depicted on the left-hand side of Figure 1.

Game G_1: Regrouping of machines and addition of relays. We regroup every machine except for $\mathcal{Z}$ from game G_0 into one machine and call it $\mathcal{S}$. We add single relays for every party, outside of $\mathcal{S}$, and one relay called $\mathcal{F}$ covering all wires between the single relays and $\mathcal{S}$.

Obviously the view of $\mathcal{Z}$ is distributed exactly as in game G_0. Figure 1 shows the transformation from G_0 to G_1 in a situation with one honest and one corrupted party.

Game G_2: Merging wires. Merge all wires between $\mathcal{F}$ and $\mathcal{S}$ into one wire. Let $\mathcal{F}$ determine recipients of messages (consisting of a tuple) from $\mathcal{S}$ by choosing the first party that occurs in the tuple. $\mathcal{S}$ determines recipients in the same way.

Messages are delivered to the same recipients as in the previous game, hence, the view of $\mathcal{Z}$ is distributed exactly as before.

[7] Throughout the paper we assume $\mathcal{Z}$ to be uniform, i.e. $\mathcal{Z}$ gets no auxiliary input.

The main difference between G_2 and the ideal execution with $\mathcal{F}_{NIKE}$ is that in G_2 the keys of honest sessions are computed using the algorithm `NIKE.SharedKey`, whereas in the ideal execution the keys are randomly chosen. This will change in the last game G_5. Next, in game G_3, we make a simple but slightly technical modification: we let $\mathcal{F}$ perform a check to determine whether it should forward a shared key coming from $\mathcal{S}$ to the requesting party. (Namely, if the other party involved in this session has not registered its public key yet, then $\mathcal{F}$ can answer this request with $\perp$ on its own.)

Game G_3: Allowing $\mathcal{F}$ to store information and make decisions. We let $\mathcal{S}$ send $(\mathsf{P}_i, \texttt{registered})$ to $\mathcal{F}$ whenever a party successfully registers a public key pk_i with $\mathcal{F}_{KR}^1$. We let $\mathcal{F}$ bounce the message back to $\mathcal{S}$ and additionally maintain a list Λ_{reg} with parties for which $\mathcal{F}$ already received such a message. Upon receiving $(\texttt{init}, \mathsf{P}_i, \mathsf{P}_j)$, if $\mathsf{P}_j \notin \Lambda_{reg}$, $\mathcal{F}$ sends $(\mathsf{P}_i, \mathsf{P}_j, \perp)$ to P_i. Else $\mathcal{F}$ relays $(\texttt{init}, \mathsf{P}_i, \mathsf{P}_j)$ to $\mathcal{S}$ and receives an answer $(\mathsf{P}_i, \mathsf{P}_j, K_{i,j})$. $\mathcal{F}$ relays $(\mathsf{P}_i, \mathsf{P}_j, K_{i,j})$ to P_i.

We have to check whether the output of P_i in G_2 is $(\mathsf{P}_i, \mathsf{P}_j, \perp)$ if and only if the output of P_i in G_3 is $(\mathsf{P}_i, \mathsf{P}_j, \perp)$. In G_2, P_i outputs $(\mathsf{P}_i, \mathsf{P}_j, \perp)$ if and only if $\perp \leftarrow \texttt{NIKE.SharedKey}(\mathsf{P}_j, \mathsf{pk}_j, \mathsf{P}_i, \mathsf{sk}_i)$. By definition of $\mathcal{F}$ in G_3, the output of a party P_i is $(\mathsf{P}_i, \mathsf{P}_j, \perp)$ if and only if $\mathsf{P}_j \notin \Lambda_{reg}$ or $\mathcal{S}$ answered with $(\mathsf{P}_i, \mathsf{P}_j, \perp)$. A missing pk_j will cause $\texttt{NIKE.SharedKey}(\mathsf{P}_j, \mathsf{pk}_j, \mathsf{P}_i, \mathsf{sk}_i)$ to output $\perp$, hence, both events together are equivalent to $\perp \leftarrow \texttt{NIKE.SharedKey}(\mathsf{P}_j, \mathsf{pk}_j, \mathsf{P}_i, \mathsf{sk}_i)$.

Game G_4: More lists and more decisions for $\mathcal{F}$. Here, we introduce two new lists, Λ_{keys} and Λ_{renew}, to $\mathcal{F}$. These lists resemble the lists used internally by $\mathcal{F}_{NIKE}$. Specifically, whenever $\mathcal{F}$ has to send a message $(\mathsf{P}_i, \mathsf{P}_j, key)$ to P_i where $key \neq \perp$, it also stores $(\{\mathsf{P}_i, \mathsf{P}_j\}, key)$ to Λ_{keys}. Whenever $\mathcal{F}$ receives a message $(\texttt{renew}, \mathsf{P}_i)$ from P_i, it adds P_i to Λ_{renew}. Whenever $\mathcal{F}$ receives a message $(\mathsf{P}_i, \texttt{registered})$ from $\mathcal{S}$, if $\mathsf{P}_i \in \Lambda_{renew}$, $\mathcal{F}$ deletes all entries $(\{\mathsf{P}_i, \cdot\}, key)$ from Λ_{keys} and removes P_i from Λ_{renew}. So far there were no modifications regarding $\mathcal{F}$'s outputs. Now upon receipt of $(\texttt{init}, \mathsf{P}_i, \mathsf{P}_j)$ with $\mathsf{P}_j \in \Lambda_{reg}$ we let $\mathcal{F}$ check the list Λ_{keys} for an entry $(\{\mathsf{P}_i, \mathsf{P}_j\}, key)$. If there is one, $\mathcal{F}$ does not relay $(\texttt{init}, \mathsf{P}_i, \mathsf{P}_j)$ to the adversary and instead returns $(\{\mathsf{P}_i, \mathsf{P}_j\}, key)$ to P_i right away.

The output of $\mathcal{F}$ is the same as in Game G_3, because any entry in Λ_{keys} was computed by $\mathcal{S}$ and therefore matches the answer of $\mathcal{S}$ to the `init` request in Game G_3.

Game G_5: Building the ideal functionality $\mathcal{F}_{NIKE}$. We now substitute all real shared keys between two honest parties in Λ_{reg} (computed via the `NIKE.SharedKey` algorithm) with random keys. Concretely, for every honest session $(\texttt{init}, \mathsf{P}_i, \mathsf{P}_j)$, for P_j in $\mathcal{F}$'s list Λ_{reg}, we let $\mathcal{F}$ determine the key for that session. First of all we prevent $\mathcal{F}$ from forwarding $(\texttt{init}, \mathsf{P}_i, \mathsf{P}_j)$ to the adversary. Next, if Λ_{keys} contains an entry $(\{\mathsf{P}_i, \mathsf{P}_j\}, K_{i,j})$, $\mathcal{F}$ sets $key = K_{i,j}$. Else $\mathcal{F}$ chooses $key \xleftarrow{\$} \{0,1\}^k$, stores $(\{\mathsf{P}_i, \mathsf{P}_j\}, key)$ and sends $(\mathsf{P}_i, \mathsf{P}_j, key)$ to P_i.

Let $\mathcal{Z}$ be a distinguishing environment between games G_4 and G_5. We use $\mathcal{Z}$ to construct an adversary $\mathcal{B}$ against `NIKE` in the CKS^+ security game. Besides playing the CKS^+ security game with its challenger $\mathcal{C}$, $\mathcal{B}$ runs $\mathcal{Z}$ and acts as a mediator between $\mathcal{Z}$ and $\mathcal{C}$.

Table 1 shows requests of $\mathcal{Z}$ and the corresponding queries in the CKS^+ security game that $\mathcal{B}$ issues to get answers for $\mathcal{Z}$'s requests. Note that $\mathcal{B}$ can embed its own challenge into the UC execution with $\mathcal{Z}$ by answering initialization requests for honest sessions from $\mathcal{Z}$ with $\mathcal{C}$'s responses to `test` queries. We omit a detailed description of $\mathcal{B}$ and briefly list what $\mathcal{B}$ has to do besides issuing the requests shown in Table 1.

- To be able to answer `lookup` requests from $\mathcal{Z}$, $\mathcal{B}$ has to keep track of all public keys.

UC requests	CKS^+ queries
$(\texttt{parameters})$	$(\texttt{parameters})$
$(\texttt{corrupt}, \mathsf{P}_i)$ + $(\texttt{register}, \mathsf{P}_i, \mathsf{pk}_i)$	$(\texttt{register corrupt user}, \mathsf{P}_i, \mathsf{pk}_i)$
$(\texttt{register}, \mathsf{P}_i)$ or $(\texttt{renew}, \mathsf{P}_i)$	$(\texttt{register honest user}, \mathsf{P}_i)$
$(\texttt{init}, \mathsf{P}_i, \mathsf{P}_j)$, corrupt session	$(\texttt{corrupt reveal}, \mathsf{P}_i, \mathsf{P}_j)$
$(\texttt{init}, \mathsf{P}_i, \mathsf{P}_j)$, honest session	$(\texttt{test}, \mathsf{P}_i, \mathsf{P}_j)$

Table 1: Corresponding queries

- When answering $(\texttt{init}, \mathsf{P}_i, \mathsf{P}_j)$ requests from $\mathcal{Z}$, $\mathcal{B}$ returns $\perp$ to $\mathcal{Z}$ if P_j has not been registered with $\mathcal{C}$ yet.

Note that according to Table 1 both $(\texttt{register}, \mathsf{P}_i)$ and $(\texttt{renew}, \mathsf{P}_i)$ requests from $\mathcal{Z}$ lead to the same request in the CKS^+ game. This is due to the fact that renewing a public key in the CKS^+ game is done by re-registering the user as honest.

Let $\mathrm{Adv}_{\mathcal{Z}}^{G_4,G_5} = |(\Pr[1 \leftarrow \mathcal{Z}|\mathcal{Z} \text{ is running in } G_5] - \Pr[1 \leftarrow \mathcal{Z}|\mathcal{Z} \text{ is running in } G_4])|$ denote the advantage of $\mathcal{Z}$ in distinguishing G_4 and G_5. By assumption, $\mathrm{Adv}_{\mathcal{Z}}^{G_4,G_5}$ is non-negligible. Let $\hat{b}$ denote the output bit of $\mathcal{Z}$ and b the bit chosen by the CKS^+ challenger $\mathcal{C}$. If $b = 0$, $\mathcal{C}$ answers $\texttt{test}$ queries with real shared keys, else the keys are randomly chosen from $\{0,1\}^k$. Thus if $b = 0$, $\mathcal{B}$ simulates G_4, and if $b = 1$, $\mathcal{B}$ simulates G_5. We let $\mathcal{B}$ output $\hat{b}$, i.e. the same bit as $\mathcal{Z}$. Hence, we have that $\mathrm{Adv}_{\mathcal{B}}^{CKS^+} = \mathrm{Adv}_{\mathcal{Z}}^{G_4,G_5}$. Clearly, as $\mathrm{Adv}_{\mathcal{Z}}^{G_4,G_5}$ is non-negligible, $\mathrm{Adv}_{\mathcal{B}}^{CKS^+}$ is non-negligible as well. This contradicts the CKS^+ security of $\texttt{NIKE}$ and thus we conclude that $\mathrm{Adv}_{\mathcal{Z}}^{G_4,G_5}$ is negligible.

It is easy to see that $\mathcal{F}$ in game G_5 behaves exactly like $\mathcal{F}_{NIKE}$, and game G_5 is equal to an ideal execution of $\texttt{NIKE}$ with $\mathcal{F}_{NIKE}$ and $\mathcal{S}$. It follows that

$$\mathrm{Exec}[\texttt{NIKE}, \mathcal{A}, \mathcal{Z}] = \mathrm{GAME}_{\mathcal{Z}}^{G_0} \approx \mathrm{GAME}_{\mathcal{Z}}^{G_5} = \mathrm{Exec}[\mathcal{F}_{NIKE}, \mathcal{S}, \mathcal{Z}],$$

where $\mathrm{GAME}_{\mathcal{Z}}^{G_0}$ (resp. $\mathrm{GAME}_{\mathcal{Z}}^{G_5}$) denotes the output of $\mathcal{Z}$ when running in game G_0 (resp. G_5). □

Remark. The hybrid functionality $\mathcal{F}_{CRS}$ is required to guarantee that the parameters for the NIKE scheme cannot be adversarially chosen. But since there is no need for the simulator to program the CRS, we can also assume $\mathcal{F}_{CRS}$ to be a *global functionality*. The global CRS functionality, denoted by $\bar{\mathcal{G}}_{CRS}$, can be directly accessed by $\mathcal{Z}$ and is a strictly weaker assumption than our functionality $\mathcal{F}_{CRS}$. A detailed description of $\bar{\mathcal{G}}_{CRS}$ can be found in [10].

Theorem 4.1 states that CKS^+ security (or any equivalent notion) of a NIKE scheme is sufficient for UC security of the scheme with respect to static adversaries. Next we show that CKS^+ security (or any equivalent notion) is also a requirement for UC-secure NIKE schemes. We recall that there is an equivalence between several flavours of game-based security notions for NIKE (see [18] or Appendix B) and it therefore suffices to show *CKS-light* security.

Theorem 4.2. *Let $\texttt{NIKE}$ be a UC-secure NIKE scheme realizing $\mathcal{F}_{NIKE}$ in the ($\mathcal{F}_{CRS}$,$\mathcal{F}_{KR}^1$)-hybrid model with respect to static corruptions. Then $\texttt{NIKE}$ is CKS-light-secure.*

Proof sketch. We will first map any *CKS-light* adversary $\mathcal{B}$ to a suitable environment $\mathcal{Z}$ that simulates $\mathcal{B}$ and translates $\mathcal{B}$'s queries in a similar way as shown in Table 1.[8] Hence, running $\mathcal{Z}$ with $\texttt{NIKE}$ provides $\mathcal{B}$ with a view as in the *CKS-light* game with $b = 0$ (i.e., with real keys). Now consider an environment $\tilde{\mathcal{Z}}$ that works like $\mathcal{Z}$, but substitutes all shared keys between honest parties

[8]Note, however, that the *CKS-light* game does not feature $\texttt{renew}$ queries. Furthermore, the number of queries considered in the *CKS-light* game is in fact more restricted than in the CKS^+ game.

provided by NIKE (or $\mathcal{F}_{NIKE}$) with random keys. Hence, running $\tilde{\mathcal{Z}}$ with NIKE provides $\mathcal{B}$ with a view as in the *CKS-light* game with $b = 1$ (i.e., with random keys). Intuitively, if $\mathcal{B}$ is a successful *CKS-light* distinguisher, then $\mathcal{B}$ can distinguish between running with $\mathcal{Z}$ or $\tilde{\mathcal{Z}}$ *in the real UC model*. However, in the ideal UC model, $\mathcal{Z}$ and $\tilde{\mathcal{Z}}$ provide $\mathcal{B}$ with identical views; hence, $\mathcal{B}$ will not be able to distinguish running with $\mathcal{Z}$ or $\tilde{\mathcal{Z}}$ in the ideal UC model. In this way, a successful $\mathcal{B}$ acts as a successful distinguisher between the real and the ideal UC model. (See Appendix F for a full proof.) □

On the UC notion of static corruption. In the proof of Theorem 4.2, we use the relatively loose definition of static corruption in the UC model (cf. [6, Section 6.7]). In fact, Theorem 4.2 would not hold if we applied a stricter (with respect to adversarial constraints) definition of static corruption. For instance, we could require that no corruptions take place after *the first* honest party receives an input. (This is in fact the default notion of static corruptions in the GNUC model of security, at least since the December 2012 update of [23].) For this notion of static corruption, we can construct counterexamples to Theorem 4.2. On the other hand, once there is only a fixed polynomial number of parties whose identities are known in advance, the environment $\mathcal{Z}$ from the proof of Theorem 4.2 can guess the two honest parties that $\mathcal{B}$ chooses and corrupt all other parties in advance. Hence, Theorem 4.2 holds even with respect to stricter notions of static corruption, once the set of possible honest parties is polynomially small.

4.2 Adaptive corruption

We consider adaptive corruption, where $\mathcal{Z}$ is now allowed to corrupt formerly honest protocol participants. To avoid trivial protocols, say that a function f (that recognize valid keys in $\mathcal{F}_{KR}^{f}$) is *nontrivial* for a given NIKE scheme iff $f(\mathsf{P}_i, \mathsf{pk}_i, \tau) = 1$ for all parameters *params* generated by $\texttt{NIKE.CommonSetup}(1^k)$ and all public keys generated by $\texttt{NIKE.KeyGen}(\mathit{params}, \mathsf{P}_i)$ along with proofs τ.

Theorem 4.3. *There is no NIKE scheme* NIKE *and function f which is nontrivial for* NIKE*, such that* NIKE *realizes $\mathcal{F}_{NIKE}$ in the ($\mathcal{F}_{CRS}$,$\mathcal{F}_{KR}^{f}$)-hybrid model with respect to adaptive corruptions.*

Proof. We specify an adversary $\mathcal{A}$ and an environment $\mathcal{Z}$ and show that there is no simulator $\mathcal{S}$ for this setup.

Let NIKE be any non-interactive key exchange scheme. W.l.o.g we assume that all secret keys have the same bit length $l = l(k)$, where k denotes the security parameter. Let NIKE.SharedKey be the shared key algorithm of NIKE that takes as inputs $(\mathsf{P}_i, \mathsf{pk}_i, \mathsf{P}_j, \mathsf{sk}_j), i \neq j$, and outputs shared keys of length k. Let $n := \lceil \frac{l}{k} \rceil + 2$. For convenience we let $\mathcal{A}$ be the dummy adversary and let $\mathcal{Z}$ mount the attack on NIKE, which is described in five steps:

1. Send (parameters) to $\mathcal{A}$ to obtain *params*, a set of system parameters.
2. Send $(\texttt{register}, \mathsf{P}_i)$ to P_i, $i = 1, \ldots, n$ and $(\texttt{init}, \mathsf{P}_i, \mathsf{P}_1)$ to P_i, $i = 2, \ldots, n$. Thus, $\mathcal{Z}$ obtains the shared keys $K_{1,i}$ between party P_1 and party P_i, $i = 2, \ldots, n$.
3. Obtain the public keys $\mathsf{pk}_1, \ldots, \mathsf{pk}_n$, corresponding to parties $\mathsf{P}_1, \ldots, \mathsf{P}_n$, from $\mathcal{A}$.
4. Send $(\texttt{corrupt}, \mathsf{P}_1)$ to $\mathcal{A}$ to learn sk_1. Abort if $\mathsf{sk}_1 \notin \{0,1\}^l$.
5. If $K_{1,i} = \texttt{NIKE.SharedKey}(\mathsf{P}_i, \mathsf{pk}_i, \mathsf{P}_1, \mathsf{sk}_1)\ \forall\, i = 2, \ldots, n$, output 1, else output 0.

In the real world $\mathcal{Z}$ will always output 1 by the correctness of NIKE and the nontriviality of f. Now let $\mathcal{S}$ be any simulator. The ideal world execution with $\mathcal{S}$ will have to proceed as follows:

1. $\mathcal{S}$ arbitrarily chooses a set of system parameters *params* and sends *params* to $\mathcal{Z}$.
2. Here $\mathcal{S}$ cannot choose the shared keys $\tilde{K}_{1,i}$, $i = 2, \ldots, n$, between the honest parties, because these are chosen by $\mathcal{F}_{NIKE}$ (and in fact unknown to $\mathcal{S}$).

3. Now $\mathcal{S}$ arbitrarily chooses public keys $\mathsf{pk}_1, \ldots, \mathsf{pk}_n$, simulating $\mathcal{F}_{KR}^f$, and sends them to $\mathcal{Z}$.
4. $\mathcal{S}$ also corrupts P_1 and learns $\tilde{K}_{1,i}$, $i = 2, \ldots, n$. $\mathcal{S}$ chooses $\mathsf{sk}_1 \in \{0,1\}^l$ and sends it to $\mathcal{Z}$.

To see what happens in step 5 we define

$$\begin{aligned} F_{\vec{\mathsf{pk}}}(\mathsf{sk}_1) : \{0,1\}^l &\longrightarrow \{0,1\}^{(n-1)k} \\ \mathsf{sk}_1 &\longmapsto (\texttt{NIKE.SharedKey}(\mathsf{P}_2, \mathsf{pk}_2, \mathsf{P}_1, \mathsf{sk}_1), \ldots, \\ &\qquad \texttt{NIKE.SharedKey}(\mathsf{P}_n, \mathsf{pk}_n, \mathsf{P}_1, \mathsf{sk}_1)) \end{aligned}$$

where $\vec{\mathsf{pk}} := (\mathsf{pk}_2, \ldots, \mathsf{pk}_n)$. $\mathcal{F}_{NIKE}$ chooses $(\tilde{K}_{1,2}, \ldots, \tilde{K}_{1,n})$ uniformly from $\{0,1\}^{(n-1)k}$, hence the probability that there exists sk_1 with

$$F_{\vec{\mathsf{pk}}}(\mathsf{sk}_1) = (\tilde{K}_{1,2}, .., \tilde{K}_{1,n})$$

is at most $\frac{2^l}{2^{(n-1)k}}$. As n was chosen such that $(n-1)k \geq l + k$, we have $\frac{2^l}{2^{(n-1)k}} \leq \frac{2^l}{2^{l+k}} \leq 2^{-k}$, which is negligible in k. It follows that $\mathcal{Z}$ will output 1 only with negligible probability. □

Remarks. (1) If the secret key depends deterministically on the public key, the attack is simpler, because after the first step $\mathcal{S}$ is committed to secret keys. After receiving the public keys, $\mathcal{Z}$ initializes a key exchange session between two honest parties and later corrupts one of them. Due to the commitment in the first step, the probability that the secret key matches the uniformly chosen shared key is negligible in the size k of the shared key.
(2) In the security models from [18], adaptive corruption corresponds to an adversary which issues a `test` query for two honest identities and additionally uses an `extract` query to get one of the secret keys. This case is excluded as trivial win for the adversary.

4.2.1 Relation to a similar result of Nielsen in secure message transfer

Nielsen's results. We have shown that security against adaptive corruptions is not possible without additional assumptions. Taking into account the results of Nielsen [27], this is not surprising. There, it is shown that there is no non-interactive protocol which realizes secure message transfer (SMT) without additional assumptions. Non-interactive SMT protocols (according to the definition of [27]) can essentially be viewed as a PKE scheme. The impossibility result is due to the fact that the simulator has to commit to a transcript of encryptions before knowing the underlying messages. Upon corruption, $\mathcal{S}$, now knowing the messages, can only hope to adjust the secret key to explain the transcript. Nielsen shows that a secret key of fixed length does not provide enough entropy to explain an unbounded number of encryptions.

Why his result does not imply ours. Regarding `NIKE`, a similar problem arises when formerly honest parties, when being corrupted, reveal an unbounded number of earlier computed shared keys. The simulator has to use the secret key to explain those shared keys. Analogous to [27] we can prove an impossibility result due to a lack of entropy in the secret key.

One could hope to conclude this directly from the result of [27] with the following argument: every `NIKE` can be used to realize SMT. Thus an impossibility result for the latter would imply Theorem 4.3. However, [27] crucially uses that the SMT protocol can transmit arbitrarily long messages between two parties. In contrast, a `NIKE` only creates fixed-length (i.e., k-bit) shared keys between each pair of parties. Such a short key cannot (in any obvious way) be used to transmit arbitrarily long messages *against adaptive corruptions*.

Therefore Nielsen's impossibility result for non-interactive SMT can not be used to directly conclude Theorem 4.3. Nevertheless our technique is strongly inspired by the idea of Nielsen, namely using an unbounded number of key exchange sessions instead of unbounded-length messages.

Generalizations of our negative result to other (non-programmable) functionalities. Theorem 4.3 shows that security against adaptive corruption is not achievable in the $(\mathcal{F}_{CRS},\mathcal{F}_{KR}^{f})$-hybrid model. The result would be even stronger if we added more powerful hybrid functionalities. We find that, similar to another result from [27], even a global random oracle functionality to which $\mathcal{Z}$ has direct access does not facilitate security against adaptive corruption. (This notion of a globally accessible functionality has been formalized in [27], but can be cast more generally in the "GUC" variant of UC [10].)

Even more, (the proof of) Theorem 4.3 would still hold in any hybrid model that, in addition to $\mathcal{F}_{CRS}$ and $\mathcal{F}_{KR}^{f}$, provides only *non-programmable* hybrid functionalities. In this context, by non-programmable we mean that the input/output behaviour of the functionality is completely independent of the simulator (note that this also includes that the simulator is not able to program the output of the functionality via scheduling of messages). This observation points us to hybrid functionalities that actually facilitate adaptive corruption. And indeed, in Section 4.4 we will see that a programmable random oracle functionality is enough to achieve adaptive UC security.

4.3 Summary of relations established so far

There is a strong relation between the game-based security notions from [12, 18] and UC security with respect to $\mathcal{F}_{NIKE}$, our functionality for non-interactive key exchange. In Section 4.1 we have shown equivalence of *CKS* security (or other equivalent notions from [18] or Appendix B) to static UC-NIKE security (w.r.t $\mathcal{F}_{NIKE}$ using $\mathcal{F}_{CRS}$ and $\mathcal{F}_{KR}$). Furthermore, Theorem 4.3 implies that *CKS* security is not enough to achieve adaptive UC-NIKE security (w.r.t $\mathcal{F}_{NIKE}$ using $\mathcal{F}_{CRS}$ and $\mathcal{F}_{KR}$). The relations between the security notions are depicted in Figure 2.

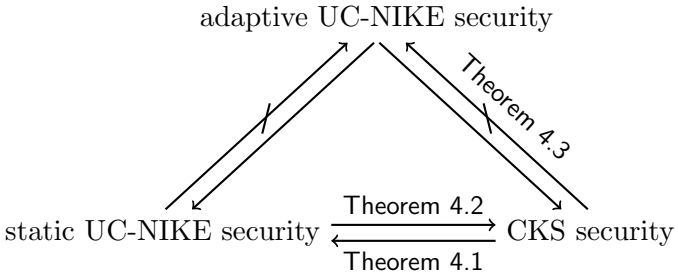

Figure 2: Relations between *CKS* security models and UC-NIKE security

4.4 Transformation to adaptively secure NIKE in the random oracle model

In Section 4.2 we have proven that, without additional assumptions, $\mathcal{F}_{NIKE}$ cannot be realized in the $(\mathcal{F}_{CRS},\mathcal{F}_{KR}^{f})$-hybrid model in the presence of adaptive adversaries. We now show how to achieve adaptive UC security if we assume the existence of a random oracle. More specifically, we show that if a NIKE scheme `NIKE` is secure in the sense of a strictly weaker ("search-based" instead of indistinguishability-based) notion of security than the notions presented in [18], and with additional allowance of re-registration of honest users and corrupt registration of previously registered honest

users, then a hash variant of NIKE securely realizes our non-interactive key exchange functionality $\mathcal{F}_{NIKE}$ in the $\mathcal{F}_{RO}$-hybrid model (see Appendix D for a description of $\mathcal{F}_{RO}$). The security model used in our reduction is denoted by *weakCKS*$^{++}$ (see Appendix B for a definition).

Definition 4.4 (Transformation to the random oracle model)**.** *Let* NIKE *be a non-interactive key exchange scheme with shared key algorithm* NIKE.SharedKey *and let* $H : \{0,1\}^* \to \{0,1\}^{k'}$ *be a hash function (viewed as a random oracle), where* k' *is a security parameter. Let* NIKE' *be a modification of the scheme* NIKE *such that its shared key algorithm,* NIKE'.SharedKey'*, is defined as*

$$\text{NIKE'.SharedKey'}(\mathsf{P}_i, \mathsf{pk}_i, \mathsf{P}_j, \mathsf{sk}_j)$$
$$\text{if } \begin{cases} \mathsf{P}_i < \mathsf{P}_j \text{ return } H(\mathsf{P}_i, \mathsf{P}_j, \text{NIKE.SharedKey}(\mathsf{P}_i, \mathsf{pk}_i, \mathsf{P}_j, \mathsf{sk}_j)) \\ \mathsf{P}_j < \mathsf{P}_i \text{ return } H(\mathsf{P}_j, \mathsf{P}_i, \text{NIKE.SharedKey}(\mathsf{P}_i, \mathsf{pk}_i, \mathsf{P}_j, \mathsf{sk}_j)) \end{cases}$$

Here we are assuming that the party identifiers come from a space with a natural ordering and that the shared key space of NIKE' *is* $\{0,1\}^{k'}$*.*

Theorem 4.5. *Let* NIKE *be a weakCKS*$^{++}$*-secure non-interactive key exchange protocol. Then the transformed protocol* NIKE' *realizes* $\mathcal{F}_{NIKE}$ *in the* $(\mathcal{F}_{CRS}, \mathcal{F}_{KR}^1, \mathcal{F}_{RO})$*-hybrid model, in the presence of adaptive adversaries.*

Proof sketch. We show that $\mathcal{Z}$ cannot distinguish the real protocol run of NIKE' with the ideal functionalities $\mathcal{F}_{CRS}$, $\mathcal{F}_{KR}^1$ and $\mathcal{F}_{RO}$, and an adversary $\mathcal{A}$, from the ideal protocol run with $\mathcal{F}_{NIKE}$ and $\mathcal{S}$, unless $\mathcal{Z}$ makes a specific type of random oracle query. Then we show that if $\mathcal{Z}$ makes such a random oracle query, we can construct an adversary against NIKE in the *weakCKS*$^{++}$ security game. By our assumption, there is no such adversary with non-negligible advantage. Thus, the probability that $\mathcal{Z}$ makes that random oracle query is negligible and therefore the real and ideal execution (with our simulator $\mathcal{S}$) cannot be distinguished by any environment $\mathcal{Z}$. (See Appendix G for a full proof.) □

References

[1] Boaz Barak, Ran Canetti, Jesper Buus Nielsen, and Rafael Pass. Universally composable protocols with relaxed set-up assumptions. In *45th Annual Symposium on Foundations of Computer Science*, pages 186–195. IEEE Computer Society Press, October 2004.

[2] Elaine Barker, Don Johnson, and Miles Smid. NIST special publication 800-56A: Recommendation for pair-wise key establishment schemes using discrete logarithm cryptography (revised), March 2007.

[3] Mihir Bellare and Phillip Rogaway. Entity authentication and key distribution. In Douglas R. Stinson, editor, *Advances in Cryptology – CRYPTO'93*, volume 773 of *Lecture Notes in Computer Science*, pages 232–249. Springer, August 1993.

[4] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd Annual Symposium on Foundations of Computer Science*, pages 136–145. IEEE Computer Society Press, October 2001.

[5] Ran Canetti. Universally composable signature, certification, and authentication. In *Proceedings of CSFW 2004*, pages 219–. IEEE Computer Society, 2004.

[6] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. Cryptology ePrint Archive, Report 2000/067, 2005. `http://eprint.iacr.org/`.

[7] Ran Canetti and Hugo Krawczyk. Analysis of key-exchange protocols and their use for building secure channels. In Birgit Pfitzmann, editor, *Advances in Cryptology – EUROCRYPT 2001*, volume 2045 of *Lecture Notes in Computer Science*, pages 453–474. Springer, May 2001.

[8] Ran Canetti and Hugo Krawczyk. Universally composable notions of key exchange and secure channels. In Lars R. Knudsen, editor, *Advances in Cryptology – EUROCRYPT 2002*, volume 2332 of *Lecture Notes in Computer Science*, pages 337–351. Springer, April / May 2002.

[9] Ran Canetti, Yehuda Lindell, Rafail Ostrovsky, and Amit Sahai. Universally composable two-party and multi-party secure computation. In *34th Annual ACM Symposium on Theory of Computing*, pages 494–503. ACM Press, May 2002.

[10] Ran Canetti, Yevgeniy Dodis, Rafael Pass, and Shabsi Walfish. Universally composable security with global setup. In Salil P. Vadhan, editor, *TCC 2007: 4th Theory of Cryptography Conference*, volume 4392 of *Lecture Notes in Computer Science*, pages 61–85. Springer, February 2007.

[11] Cagatay Capar, Dennis Goeckel, Kenneth G. Paterson, Elizabeth A. Quaglia, Don Towsley, and Murtaza Zafer. Signal-flow-based analysis of wireless security protocols. *Inf. Comput.*, 226:37–56, 2013.

[12] David Cash, Eike Kiltz, and Victor Shoup. The twin Diffie-Hellman problem and applications. In Nigel P. Smart, editor, *Advances in Cryptology – EUROCRYPT 2008*, volume 4965 of *Lecture Notes in Computer Science*, pages 127–145. Springer, April 2008.

[13] Ivan Damgård, Jesper Buus Nielsen, and Daniel Wichs. Universally composable multiparty computation with partially isolated parties. In Omer Reingold, editor, *TCC 2009: 6th Theory of Cryptography Conference*, volume 5444 of *Lecture Notes in Computer Science*, pages 315–331. Springer, March 2009.

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

[15] Yevgeniy Dodis, Jonathan Katz, Adam Smith, and Shabsi Walfish. Composability and on-line deniability of authentication. In Omer Reingold, editor, *TCC 2009: 6th Theory of Cryptography Conference*, volume 5444 of *Lecture Notes in Computer Science*, pages 146–162. Springer, March 2009.

[16] Régis Dupont and Andreas Enge. Provably secure non-interactive key distribution based on pairings. *Discrete Applied Mathematics*, 154(2):270–276, 2006.

[17] Roger Fischlin and Claus-Peter Schnorr. Stronger security proofs for RSA and Rabin bits. *Journal of Cryptology*, 13(2):221–244, 2000.

[18] Eduarda S. V. Freire, Dennis Hofheinz, Eike Kiltz, and Kenneth G. Paterson. Non-interactive key exchange. In *PKC 2013: 16th International Workshop on Theory and Practice in Public Key Cryptography*, Lecture Notes in Computer Science, pages 254–271. Springer, 2013. doi: 10.1007/978-3-642-36362-7_17.

[19] Eduarda S. V. Freire, Dennis Hofheinz, Kenneth G. Paterson, and Christoph Striecks. Programmable hash functions in the multilinear setting. Lecture Notes in Computer Science, pages 513–530. Springer, August 2013. doi: 10.1007/978-3-642-40041-4_28.

[20] Rosario Gennaro, Shai Halevi, Hugo Krawczyk, Tal Rabin, Steffen Reidt, and Stephen D. Wolthusen. Strongly-resilient and non-interactive hierarchical key-agreement in MANETs. In Sushil Jajodia and Javier López, editors, *ESORICS 2008: 13th European Symposium on Research in Computer Security*, volume 5283 of *Lecture Notes in Computer Science*, pages 49–65. Springer, October 2008.

[21] Shafi Goldwasser, Silvio Micali, and Ronald L. Rivest. A "paradoxical" solution to the signature problem (extended abstract). In *25th Annual Symposium on Foundations of Computer Science*, pages 441–448. IEEE Computer Society Press, October 1984.

[22] Dennis Hofheinz and Eike Kiltz. The group of signed quadratic residues and applications. In Shai Halevi, editor, *Advances in Cryptology – CRYPTO 2009*, volume 5677 of *Lecture Notes in Computer Science*, pages 637–653. Springer, August 2009.

[23] Dennis Hofheinz and Victor Shoup. GNUC: A new universal composability framework. Cryptology ePrint Archive, Report 2011/303, 2011. `http://eprint.iacr.org/`.

[24] Dennis Hofheinz, Jörn Müller-Quade, and Rainer Steinwandt. Initiator-resilient universally composable key exchange. In Einar Snekkenes and Dieter Gollmann, editors, *ESORICS 2003: 8th European Symposium on Research in Computer Security*, volume 2808 of *Lecture Notes in Computer Science*, pages 61–84. Springer, October 2003.

[25] Markus Jakobsson, Kazue Sako, and Russell Impagliazzo. Designated verifier proofs and their applications. In Ueli M. Maurer, editor, *Advances in Cryptology – EUROCRYPT'96*, volume 1070 of *Lecture Notes in Computer Science*, pages 143–154. Springer, May 1996.

[26] Dafna Kidron and Yehuda Lindell. Impossibility results for universal composability in public-key models and with fixed inputs. *Journal of Cryptology*, 24(3):517–544, July 2011.

[27] Jesper Buus Nielsen. Separating random oracle proofs from complexity theoretic proofs: The non-committing encryption case. In Moti Yung, editor, *Advances in Cryptology – CRYPTO 2002*, volume 2442 of *Lecture Notes in Computer Science*, pages 111–126. Springer, August 2002.

[28] Kenneth G. Paterson and Sriramkrishnan Srinivasan. On the relations between non-interactive key distribution, identity-based encryption and trapdoor discrete log groups. *Des. Codes Cryptography*, 52(2):219–241, 2009.

[29] Ryuichi Sakai, Kiyoshi Ohgishi, and Masao Kasahara. Cryptosystems based on pairing. In *SCIS 2000*, Okinawa, Japan, January 2000.

[30] Victor Shoup. On formal models for secure key exchange. Technical Report RZ 3120, IBM, 1999.

A Key registration for the factoring-based NIKE from [18]

In this section, we recall the factoring-based NIKE scheme from [18] and argue why it constitutes a secure realization of $\mathcal{F}_{NIKE}$ in the $\mathcal{F}_{KR}^{f}$-hybrid model for a suitable (nontrivial) "validity" function f. We start with a bit of notation taken almost verbatim from [18].

The factoring assumption. Let $n(k)$ be a function and δ a constant with $0 \leq \delta < 1/2$. Let $\texttt{RSAgen}$ 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 [22] (see also [21, 17]) 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 recognizable given only N as input.

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

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

where $N \xleftarrow{\$} \texttt{RSAgen}(1^k)$, and $X \xleftarrow{\$} \mathbb{QR}_N^+$ and $R \xleftarrow{\$} \{0,1\}^k$ are chosen uniformly.

Chameleon hashing. A chameleon hash function $\mathrm{ChamH} : \mathcal{D} \times \mathcal{R}_{\mathrm{Cham}} \to \mathcal{I}$, 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 $\texttt{Cham.KeyGen}(1^k)$. 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 messages m induce the same probability distribution on $\mathrm{ChamH}_{\mathsf{hk}}(m, r)$ for r chosen uniformly at random.

The NIKE scheme. Let $\mathrm{ChamH} : \{0,1\}^* \times \mathcal{R}_{\mathrm{Cham}} \to \mathbb{Z}_{2^k}$ be a chameleon hash function. Now consider the following scheme $\mathrm{NIKE}_{\text{fac-int}}$:

$\texttt{NIKE.CommonSetup}(1^k)$
$(N, P, Q) \xleftarrow{\$} \texttt{RSAgen}(1^k)$
$g, u_0, u_1, u_2 \xleftarrow{\$} \mathbb{QR}_N^+$ with $\langle g \rangle = \mathbb{QR}_N^+$
$\mathsf{hk}, \mathsf{ck} \xleftarrow{\$} \texttt{Cham.KeyGen}(1^k)$
$params \leftarrow (N, g, u_0, u_1, u_2, \mathsf{hk})$
Return $params$

$\texttt{NIKE.KeyGen}(params, \mathsf{P}_i)$
$x \xleftarrow{\$} \mathbb{Z}_{\lfloor N/4 \rfloor}$; $r \xleftarrow{\$} \mathcal{R}_{\text{Cham}}$
$Z \leftarrow g^{x \cdot 2^{3k}}$; $t \leftarrow \text{ChamH}_{\mathsf{hk}}(Z || \mathsf{P}_i; r)$
$Y \leftarrow u_0 u_1^t u_2{}^{t^2}$; $X \leftarrow Y^x$
$\mathsf{pk} \leftarrow (Z, X, r)$; $\mathsf{sk} \leftarrow x$
Return $(\mathsf{pk}, \mathsf{sk})$

$\texttt{NIKE.SharedKey}(\mathsf{P}_i, \mathsf{pk}_i, \mathsf{P}_j, \mathsf{sk}_j)$
Parse $\mathsf{pk}_i =: (Z_i, X_i, r_i) \in \mathbb{QR}_N^+ \times \mathbb{QR}_N^+ \times \mathcal{R}_{\text{Cham}}$ and $\mathsf{sk}_j =: x_j \in \mathbb{Z}_{\lfloor N/4 \rfloor}$
Return $\text{BBS}_N(Z_i^{x_j \cdot 2^{2k}})$

Consistent public keys and the security of scheme. We first cite a result from [18] that shows that the NIKE scheme $\texttt{NIKE}_{\text{fac-int}}$ is *CKS-light*-secure (and thus *CKS*$^+$-secure), assuming that an adversary only registers public keys that are *consistent*. A public key $\mathsf{pk} = (Z, X, r)$ is consistent if there is an exponent x with

$$Z = g^{x \cdot 2^{3k}}, \qquad X = (u_0 u_1^t u_2^{t^2})^x \qquad \text{for } t = \text{ChamH}_{\mathsf{hk}}(Z || \mathsf{P}_i; r). \tag{1}$$

Theorem A.1 ([18])**.** *Under the generalized Riemann hypothesis and the factoring assumption relative to* RSAgen*, and assuming that the chameleon hash function ChamH is collision-resistant, the scheme* $\text{NIKE}_{\text{fac-int}}$ *is CKS*$^+$*-secure against all adversaries that only register consistent (in the sense above) public keys.*

Our goal is now to formalize the notion of consistent public keys (as in (1)) using our key registration functionality $\mathcal{F}_{KR}^f$ for a suitable f. Concretely, observe that an exponent x satisfying (1) is a witness of consistency for a given public key pk. Hence, if we set

$$f(\mathsf{P}_i, \mathsf{pk}, \tau) = 1 \quad :\Longleftrightarrow \quad Z = g^{\tau \cdot 2^{3k}} \text{ and } X = (u_0 u_1^t u_2^{t^2})^\tau, \tag{2}$$

where pk is parsed as (Z, X, r) and $t = \text{ChamH}_{\mathsf{hk}}(Z || \mathsf{P}_i; r)$, then $\mathcal{F}_{KR}^f$ allows precisely consistent public keys to be registered. (There is one subtlety: $\mathcal{F}_{KR}^f$ is not aware of the public parameters *params*, and in particular not of hk. This can be resolved by including *params* in public keys pk registered with $\mathcal{F}_{KR}^f$; each party will then of course have to check that the value of *params* is the same as provided by $\mathcal{F}_{CRS}$.)

Using a combination of Theorem A.1 and a slight adaptation of Theorem 4.1,[9] we then obtain:

Theorem A.2 ([18])**.** *Assume the generalized Riemann hypothesis, that factoring is hard relative to* RSAgen*, and that the chameleon hash function ChamH is collision-resistant. Then* $\text{NIKE}_{\text{fac-int}}$ *realizes* $\mathcal{F}_{NIKE}$ *in the* $(\mathcal{F}_{KR}^f, \mathcal{F}_{CRS})$*-hybrid model for the function f from (2) against static corruptions.*

B Game-based security notions for NIKE

Review of existing security notions

Following a game-based notion of security for non-interactive key exchange schemes from [12], several security definitions for NIKE were stated in [18]: the *CKS-light*, *CKS*, *CKS-heavy* and

[9] Recall that Theorem 4.1 assumes a key registration functionality $\mathcal{F}_{KR}^1$ that accepts all registered keys. However, the same proof also works for general f if the NIKE scheme is *CKS*$^+$ secure in a setting in which only consistent (w.r.t. f) keys are registered.

m-CKS-heavy security models. Their strongest model, the *m-CKS-heavy* security model, covers all possible types of queries that an adversary can make in any of those models. In this paper we always refer to the *dishonest key registration* (*DKR* in [18]) setting, where an adversary against a NIKE scheme is allowed to register dishonestly generated public keys for which it does not have to know the corresponding secret keys.

As all game-based security notions used in our paper are derived from the *m-CKS-heavy* security model (with possible re-registration of keys), for ease of comparison, we present here the *m-CKS-heavy* model.

In [18], the *m-CKS-heavy* security notion for non-interactive key exchange schemes is defined in terms of a game between an adversary $\mathcal{B}$ and a challenger $\mathcal{C}$. First, $\mathcal{B}$ obtains a set of system parameters, *params*, from $\mathcal{C}$, which is obtained by running $params \xleftarrow{\$} \texttt{NIKE.CommonSetup}(1^k)$. $\mathcal{C}$ then randomly chooses a bit b and answers the following oracle queries from $\mathcal{B}$:

($\texttt{register honest user}, \mathsf{ID}$). Challenger $\mathcal{C}$ generates a public key/secret key pair by running $(\mathsf{pk}, \mathsf{sk}) \xleftarrow{\$} \texttt{NIKE.KeyGen}(params, \mathsf{ID})$, records $(honest, \mathsf{ID}, \mathsf{pk}, \mathsf{sk})$ and sends pk to $\mathcal{B}$.

($\texttt{register corrupt user}, \mathsf{ID}$). $\mathcal{C}$ records $(corrupt, \mathsf{ID}, \mathsf{pk}, \perp)$, overwriting any existing entry for this ID.

($\texttt{extract}, \mathsf{ID}$). $\mathcal{C}$ looks for an entry of the form $(honest, \mathsf{ID}, \mathsf{pk}, \mathsf{sk})$ and returns sk to $\mathcal{B}$.

($\texttt{honest reveal}, \mathsf{ID}_i, \mathsf{ID}_j$). Here $\mathcal{B}$ supplies two identities ID_i and ID_j, both registered as honest. $\mathcal{C}$ obtains the shared key between ID_i and ID_j by running $K_{i,j} \leftarrow \texttt{NIKE.SharedKey}(\mathsf{ID}_i, \mathsf{pk}_i, \mathsf{ID}_j, \mathsf{sk}_j)$ and returns $K_{i,j}$ to $\mathcal{B}$.

($\texttt{corrupt reveal}, \mathsf{ID}_i, \mathsf{ID}_j$). Here $\mathcal{B}$ supplies one honest and one corrupt identity. $\mathcal{C}$ obtains the shared key between ID_i and ID_j by running $K_{i,j} \leftarrow \texttt{NIKE.SharedKey}(\mathsf{ID}_i, \mathsf{pk}_i, \mathsf{ID}_j, \mathsf{sk}_j)$ if ID_j is honest or $K_{i,j} \leftarrow \texttt{NIKE.SharedKey}(\mathsf{ID}_j, \mathsf{pk}_j, \mathsf{ID}_i, \mathsf{sk}_i)$ if ID_i is honest. $\mathcal{C}$ returns $K_{i,j}$ to $\mathcal{B}$.

($\texttt{test}, \mathsf{ID}_i, \mathsf{ID}_j$). $\mathcal{B}$ supplies two honest identities. $\mathcal{C}$ computes the corresponding shared key by running $K_{i,j} \leftarrow \texttt{NIKE.SharedKey}(\mathsf{ID}_i, \mathsf{pk}_i, \mathsf{ID}_j, \mathsf{sk}_j)$ if $b = 0$, or $K_{i,j} \xleftarrow{\$} \mathcal{SHK}$ if $b = 1$. $\mathcal{C}$ returns $K_{i,j}$ to $\mathcal{B}$.

The identities ID belong to an identity space $\mathcal{IDS}$ and are merely used to track which public keys are associated with which users. Also, $\mathcal{B}$ is allowed to make an arbitrary number of queries. We assume that $\mathcal{C}$ maintains a list of shared keys and sends the same shared key for a pair of identities if $\mathcal{B}$ makes the same `test` query again. To avoid trivial wins, $\mathcal{B}$ is not allowed to issue an `honest reveal` query and a `test` query on the same pair of identities. For the same reason, $\mathcal{B}$ is not allowed to issue an `extract` query on any identity involved in a `test` query, and vice versa. $\mathcal{B}$ is also not allowed to make a `register corrupt user` query on an identity which has already been registered as honest. $\mathcal{B}$ wins the game if it outputs a bit $\hat{b} = b$.

The advantage of $\mathcal{B}$ in the *m-CKS-heavy* security game is defined as:

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

where q_H, q_C, q_E, q_{HR}, q_{CR} and q_T are the numbers of `register honest user`, `register corrupt user`, `extract`, `honest reveal`, `corrupt reveal` and `test` queries made by $\mathcal{B}$. Informally, a NIKE scheme is *m-CKS-heavy-secure* if there is no polynomial-time adversary that makes at most q_H `register honest user` queries, etc., having non-negligible advantage in k. In our security reductions, we will w.l.o.g. restrict to adversaries that make exactly as many queries of each type as they are allowed, since we can easily transform every $(q_H, q_C, q_E, q_{HR}, q_{CR}, q_T)$-bounded adversary into one that issues exactly $(q_H, q_C, q_E, q_{HR}, q_{CR}, q_T)$ queries.

In Table 2 we summarize the differences between the *CKS-light*, *CKS*, *CKS-heavy* and *m-CKS-heavy* security notions by specifying the maximum number of queries which $\mathcal{B}$ is allowed to issue.

Security notion	`reg.hon.`	`reg.corr.`	`extract`	`hon.rev.`	`corr.rev.`	`test`
CKS-light	2	$*$	0	0	$*$	1
CKS	$*$	$*$	0	0	$*$	$*$
CKS-heavy	$*$	$*$	$*$	$*$	$*$	1
m-CKS-heavy	$*$	$*$	$*$	$*$	$*$	$*$

Table 2: Allowed number of queries for an adversary in the game-based security definitions for NIKE from [18]. $*$ means that $\mathcal{B}$ is allowed to make an arbitrary number of queries.

Theorem B.1 (Theorem 1 from [18])**.** *The CKS-light, CKS, CKS-heavy and m-CKS-heavy security models are all polynomially equivalent.*

Allowing re-registration of honest users and corrupt registration of previously registered honest users

The security notions from [18] capture several types of queries that an adversary can make. However, those models still have some shortcomings. For instance, they do not capture re-registration of honest users. Concretely, an adversary in those models is not allowed to issue (`register honest user`, ID) multiple times for the same ID and obtain a new public key pk upon each such request. This is often not realistic, e.g. when we have to work with public key/secret key pairs that have a lifetime and expiration date. So a protocol participant should be able to perform what we will call an *honest re-registration*.

Moreover none of those security models allow an adversary to register previously registered honest users as corrupt users. In this paper we augment the security notions from [18] with honest re-registration and corrupt registration of honest users and throughout the paper we add $^{++}$ to the notation to represent the augmented versions of the models from [18] including those two types of re-registration. We add a single 'plus', $^{+}$, to the notation if the models are only augmented with honest re-registration of honest users. We prove our strongest model, the *m-CKS-heavy*$^{++}$, to be polynomially equivalent to the weakest model from [18], the *CKS-light* model. Then, it follows that all security models, *with or without* the types of re-registration mentioned above, are also polynomially equivalent. Let us first introduce the *m-CKS-heavy*$^{++}$ security model.

The *m-CKS-heavy*$^{++}$ security model. We modify the *m-CKS-heavy* security notion from [18] by allowing an adversary $\mathcal{A}$ to make multiple (`register honest user`, ID) queries for the same ID. Upon each such request, $\mathcal{C}$ runs $(\mathsf{pk}, \mathsf{sk}) \xleftarrow{\$}$ `NIKE.KeyGen`$(params, \mathsf{ID})$ to obtain a new public key/secret key pair and returns pk to $\mathcal{A}$. $\mathcal{C}$ then overwrites any existing entry $(honest, \mathsf{ID}, \mathsf{pk}, \mathsf{sk})$ for ID in its list. Additionally, $\mathcal{A}$ is allowed to register previously registered honest users as corrupt users by issuing a (`register corrupt user`, ID) query to its challenger $\mathcal{C}$. The user will from then on be considered as corrupt. Here $\mathcal{C}$ deletes any existing entry for ID of the form $(honest, \mathsf{ID}, \mathsf{pk}, \mathsf{sk})$ and adds a new entry $(corrupt, \mathsf{ID}, \mathsf{pk}, \perp)$ to its list. The adversary's job in the *m-CKS-heavy*$^{++}$ game is the same as in the *m-CKS-heavy* game: $\mathcal{A}$ has to guess whether $\mathcal{C}$ answers `test` queries with the real shared key, computed via the `NIKE.SharedKey` algorithm using the current public key of one of the users involved in the query and the current secret key of the other user, or with random keys from the shared key space $\mathcal{SHK}$.

Here, we loosen the restriction required to avoid trivial wins in the *m-CKS-heavy* model. We allow an adversary against a NIKE scheme to make `test` queries on a pair of honest identities

even if an `honest reveal` query on the same pair of identities occurred before (as long as one of the users has been re-registered since the `test` query was made), and vice versa. We also allow an adversary to make `extract` queries on users involved in `test` queries as long as that user has been re-registered since the `test` query was made (the same is valid in the other direction). Notice that in order to avoid an adversary to trivially win the security game, if the challenge bit equals 1, whenever the adversary makes a `test` query on the same pair of identities $(\mathsf{ID}_i, \mathsf{ID}_j)$, the challenger must respond with a new random value if at least one of the honest identities has been re-registered since the last `test` query on $(\mathsf{ID}_i, \mathsf{ID}_j)$ was made.

Let b denote the challenge bit chosen by a challenger $\mathcal{C}$ in the *m-CKS-heavy*$^{++}$ security game and $\hat{b}$ the guess of $\mathcal{A}$. Then the advantage of an *m-CKS-heavy*$^{++}$ adversary $\mathcal{A}$ is defined as

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

We *abuse notation* here and let q_H denote the number of honest users registered by $\mathcal{A}$ and q_R the number of total honest re-registrations made by $\mathcal{A}$. As in the *m-CKS-heavy* model, here q_C, q_E, q_{HR}, q_{CR} and q_T are the numbers of `register corrupt user`, `extract`, `honest reveal`, `corrupt reveal` and `test` queries made by $\mathcal{A}$.

Remark [loosening restrictions on other security notions]: We highlight here that we also appropriately loosen restrictions on other security models. As an example, we allow an adversary in the augmented versions of the *CKS-light* or *CKS-heavy* security models to make multiple `test` queries (instead of one), *but* on the same pair of identities involved in the first `test` query and *only if* at least one of these identities has been re-registered since the last `test` query was made. Again here, if $b = 1$, the challenger chooses a new random value to answer the new `test` query.

Theorem B.2 (*m-CKS-heavy*$^{++}$ $\Leftrightarrow$ *CKS-light*)**.** *The m-CKS-heavy*$^{++}$ *and CKS-light security models are polynomially equivalent. More specifically, for any NIKE scheme* `NIKE`*, we have that:*

- *for any adversary $\mathcal{B}$ against* `NIKE` *in the CKS-light game, there is an adversary $\mathcal{A}$ that breaks* `NIKE` *in the m-CKS-heavy*$^{++}$ *game with*

$$Adv_{\mathcal{A}}^{\textit{m-CKS-heavy}^{++}}(k, 2, q_C, 0, 0, q_{CR}, 1, 0) = Adv_{\mathcal{B}}^{\textit{CKS-light}}(k, q_C, q_{CR}),$$

- *for any adversary $\mathcal{A}$ against* `NIKE` *in the m-CKS-heavy*$^{++}$ *game, there is an adversary $\mathcal{B}$ that breaks* `NIKE` *in the CKS-light game with*

$$Adv_{\mathcal{B}}^{\textit{CKS-light}}(k, q_C, q'_{CR}) \geq 2Adv_{\mathcal{A}}^{\textit{m-CKS-heavy}^{++}}(k, q_H, q_C, q_E, q_{HR}, q_{CR}, q_T, q_R)/q_T q_H^2 (q_R + 1)^2,$$

where $q'_{CR} \leq q_{CR}$.

Proof. Clearly, security in the sense of the *m-CKS-heavy*$^{++}$ model implies security in the sense of the *CKS-light* model, since the latter model is a limited case of the former.

Here we prove the second reduction, namely that if a NIKE scheme `NIKE` is secure in the *CKS-light* model, then it is also secure in the *m-CKS-heavy*$^{++}$ model. We assume that there is an adversary $\mathcal{A}$ that breaks a NIKE scheme in the *m-CKS-heavy*$^{++}$ model with advantage $\mathrm{Adv}_{\mathcal{A}}^{\textit{m-CKS-heavy}^{++}}(k, q_H, q_C, q_E, q_{HR}, q_{CR}, q_T, q_R)$.

The hybrid argument technique. We consider a sequence of games $G_0, G_1, \ldots, G_{q_T}$, all defined over the same probability space. Starting with the actual adversarial game G_0, with respect to an adversary $\mathcal{A}$ in the *m-CKS-heavy*$^{++}$ model when $b = 1$, we make slight modifications between successive games, in such a way that $\mathcal{A}$'s view is indistinguishable among the games.

Game G_0: actual adversarial game when $b = 1$. This is the original game as defined in the *m-CKS-heavy*$^{++}$ model when $b = 1$. All `test` queries will be answered with randomly chosen values from the shared key space $\mathcal{SHK}$.

Game G_ι ($1 \leq \iota \leq q_T - 1$): hybrid games. This game is identical to game $G_{\iota-1}$, except that the ι-th `test` query, say on a pair of honest users ID_i, ID_j, is answered with the actual real shared key $K_{i,j}$, between those users.

Game G_{q_T}: actual adversarial game when $b = 0$. This is the original game as defined in the *m-CKS-heavy*$^{++}$ model when $b = 0$. All `test` queries will be answered with the current real shared keys (computed via the `NIKE.SharedKey` algorithm) associated to the two users involved in each query.

Let $\mathcal{A}(G_\iota)$ denote adversary $\mathcal{A}$ playing game G_ι. We see that $\mathcal{A}$ can distinguish games G_0 and G_{q_T} with advantage

$$\mathrm{Adv}_{\mathcal{A}}^{\textit{m-CKS-heavy}^{++}}(k, q_H, q_C, q_E, q_{HR}, q_{CR}, q_T, q_R) = \left|\Pr[\mathcal{A}(G_0) = 1] - \Pr[\mathcal{A}(G_{q_T}) = 1]\right|.$$

Note that G_ι and $G_{\iota+1}$ ($0 \leq \iota < q_T$) differ in only one single `test` query. According to the *hybrid argument*, if $\mathcal{A}$ can distinguish between G_0 and G_{q_T} with non-negligible probability, then for some $0 \leq \iota < q_T$ it can also distinguish G_ι and $G_{\iota+1}$ with non-negligible probability. We show that if this is the case, then we can construct an adversary $\mathcal{B}$ in the *CKS-light* model with advantage related to $\mathcal{A}$'s advantage by a polynomial factor.

The idea. In order to simulate the *m-CKS-heavy*$^{++}$ game for $\mathcal{A}$, $\mathcal{B}$ picks $\iota \xleftarrow{\$} \{0, \ldots, q_T - 1\}$, guessing that $\mathcal{A}$ can distinguish games G_ι and $G_{\iota+1}$. Then it guesses which users will be involved in the $(\iota + 1)$-st `test` query made by $\mathcal{A}$ and also how often these users will be re-registered by $\mathcal{A}$, before issuing that `test` query.

$\mathcal{B}$ 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, computes $params \xleftarrow{\$} \texttt{NIKE.CommonSetup}(1^k)$ and gives $params$ to $\mathcal{B}$. $\mathcal{C}$ then takes a 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 honest users registered by $\mathcal{B}$ in the course of its attack and let q_R and q_T be bounds on the number of total honest re-registration queries and `test` queries made by $\mathcal{A}$. W.l.o.g., we assume that $\mathcal{A}$ does not make the same `test` query more than once if both users involved have the same key pair as when the previous `test` query was made. $\mathcal{B}$ chooses an index $\iota \xleftarrow{\$} \{0, \ldots, q_T - 1\}$, two distinct indices $I, J \xleftarrow{\$} \{1, \ldots, q_H\}$ and two other integers $\tilde{q}_{R_I}, \tilde{q}_{R_J} \xleftarrow{\$} \{0, \ldots, q_R\}$. Let q_{R_I}, q_{R_J} be the number of honest re-registrations of the I-th and J-th distinct registered honest users, respectively, issued by $\mathcal{A}$, before the $(\iota+1)$-st `test` query is made. $\mathcal{B}$ gives $params$ to $\mathcal{A}$ and answers queries to $\mathcal{A}$ in the following manner:

(`register honest user`, ID) If ID is the I-th or J-th distinct user involved in such a query, $\mathcal{B}$ sets $\mathsf{ID}_I = \mathsf{ID}$ or $\mathsf{ID}_J = \mathsf{ID}$ as appropriate. Moreover, if this is the $(\tilde{q}_{R_I} + 1)$-st (resp. $(\tilde{q}_{R_J} + 1)$-st) request of this type for ID_I (resp. ID_J), $\mathcal{B}$ adds ID_I (resp. ID_J) to a list Λ_T. $\mathcal{B}$ then makes the same query to $\mathcal{C}$, which obtains $(\mathsf{pk}, \mathsf{sk}) \xleftarrow{\$} \texttt{NIKE.KeyGen}(params, \mathsf{ID})$, records $(honest, \mathsf{ID}, \mathsf{pk}, \mathsf{sk})$, and returns pk to $\mathcal{B}$. $\mathcal{B}$ gives pk to $\mathcal{A}$.

In any other case, $\mathcal{B}$ obtains $(\mathsf{pk}, \mathsf{sk}) \xleftarrow{\$} \texttt{NIKE.KeyGen}(\mathit{params}, \mathsf{ID})$ and sends pk to $\mathcal{A}$. $\mathcal{B}$ stores $(\mathit{honest}, \mathsf{ID}, \mathsf{pk}, \mathsf{sk})$ in a list Λ_{hon}, overwriting any existing entry for ID. Also, if this is the $(\tilde{q}_{R_I} + 2)$-nd (resp. $(\tilde{q}_{R_J} + 2)$-nd) query of this type for ID_I (resp. ID_J), $\mathcal{B}$ removes ID_I (resp. ID_J) from Λ_T.

($\texttt{register corrupt user}, \mathsf{ID}, \mathsf{pk}$): $\mathcal{B}$ aborts if $\mathsf{ID} \in \{\mathsf{ID}_I, \mathsf{ID}_J\}$. If $\mathsf{ID} \notin \{\mathsf{ID}_I, \mathsf{ID}_J\}$, this means that $\mathcal{B}$ never made a ($\texttt{register honest user}, \mathsf{ID}$) query to $\mathcal{C}$. So $\mathcal{B}$ forwards the `register corrupt user` query to $\mathcal{C}$, which will record $(\mathit{corrupt}, \mathsf{ID}, \mathsf{pk}, \perp)$. $\mathcal{B}$ then checks if ID is in Λ_{hon} and if so, deletes the entry for that user. Additionally, $\mathcal{B}$ stores $(\mathit{corrupt}, \mathsf{ID}, \mathsf{pk}, \perp)$ in a list Λ_{cor}.

($\texttt{extract}, \mathsf{ID}$): If $\mathsf{ID} \notin \Lambda_T$, $\mathcal{B}$ finds $(\mathit{honest}, \mathsf{ID}, \mathsf{pk}, \mathsf{sk})$ in Λ_{hon} and gives sk to $\mathcal{A}$. Otherwise, if $\mathsf{ID} \in \Lambda_T$, $\mathcal{B}$ aborts the simulation.

($\texttt{corrupt reveal}, \mathsf{ID}_i, \mathsf{ID}_j$): Here $\mathcal{A}$ supplies two identities ID_i and ID_j, where either ID_i or ID_j is an honest user. W.l.o.g., let us assume that ID_j is the honest user. $\mathcal{B}$ checks if $\mathsf{ID}_j \in \Lambda_{\text{hon}}$. If $\mathsf{ID}_j \notin \Lambda_{\text{hon}}$ (that means that $\mathsf{ID}_j \in \Lambda_T$), $\mathcal{B}$ makes the same query to $\mathcal{C}$, obtaining $K_{i,j}$, the shared key between ID_i and ID_j. $\mathcal{B}$ returns this value to $\mathcal{A}$. Now, if $\mathsf{ID}_j \in \Lambda_{\text{hon}}$, $\mathcal{B}$ finds sk_j, then it finds pk_i in Λ_{cor} and computes $K_{i,j} \leftarrow \texttt{NIKE.SharedKey}(\mathsf{ID}_i, \mathsf{pk}_i, \mathsf{ID}_j, \mathsf{sk}_j)$. $\mathcal{B}$ then returns $K_{i,j}$ to $\mathcal{A}$.

($\texttt{honest reveal}, \mathsf{ID}_i, \mathsf{ID}_j$): Here $\mathcal{A}$ supplies two identities ID_i and ID_j, both registered as honest users. If $\{\mathsf{ID}_i, \mathsf{ID}_j\} \subseteq \Lambda_T$, $\mathcal{B}$ aborts. Otherwise, at least one of the identities must be in Λ_{hon}. Without loss of generality, assume that $\mathsf{ID}_i \in \Lambda_{\text{hon}}$. Now $\mathcal{B}$ finds the secret key sk_i in Λ_{hon}, then computes $K_{i,j} \leftarrow \texttt{NIKE.SharedKey}(\mathsf{ID}_j, \mathsf{pk}_j, \mathsf{ID}_i, \mathsf{sk}_i)$ and returns $K_{i,j}$ to $\mathcal{A}$.

($\texttt{test}, \mathsf{ID}_i, \mathsf{ID}_j$): The way $\mathcal{B}$ answers `test` queries depends on the number of such queries issued by $\mathcal{A}$.

- the first ι `test` queries, $\mathcal{B}$ answers with the actual shared keys associated to the corresponding users involved in those queries. In order to do this, $\mathcal{B}$ checks if $\{\mathsf{ID}_i, \mathsf{ID}_j\} \subseteq \Lambda_T$. If so, it aborts. Otherwise, at least one of the identities must be in Λ_{hon}. W.l.o.g., assume that $\mathsf{ID}_i \in \Lambda_{\text{hon}}$. Now $\mathcal{B}$ retrieves the secret key sk_i from Λ_{hon}, then computes $K_{i,j} \leftarrow \texttt{NIKE.SharedKey}(\mathsf{ID}_j, \mathsf{pk}_j, \mathsf{ID}_i, \mathsf{sk}_i)$ and returns $K_{i,j}$ to $\mathcal{A}$.
- if this is the $(\iota + 1)$-st `test` query, $\mathcal{B}$ checks if $\{\mathsf{ID}_i, \mathsf{ID}_j\} \in \Lambda_T$. If not, it aborts. If $\{\mathsf{ID}_i, \mathsf{ID}_j\} \subseteq \Lambda_T$, $\mathcal{B}$ makes the same `test` query to $\mathcal{C}$, receiving a value α. $\mathcal{B}$ gives α to $\mathcal{A}$.
- all other `test` queries $\mathcal{B}$ answers to $\mathcal{A}$ with random values from $\mathcal{SHK}$. Here $\mathcal{B}$ responds with a new random key every time $\mathcal{A}$ makes such a query for the same pair of identities. (Remember that we are assuming that $\mathcal{A}$ does not make the same `test` query more than once if each identity involved has the same public/private key pair since the previous test query was made.)

This completes the description of $\mathcal{B}$'s simulation. Whenever $\mathcal{A}$ outputs a bit $\hat{b}$, $\mathcal{B}$ outputs the same bit. Now, if α is the actual shared key computed via the `NIKE.SharedKey` algorithm and using as inputs the current secret key of one the users involved in the $(\iota+1)$-st `test` query and the current public key of the other user, then $\mathcal{A}$ was playing game $G_{\iota+1}$. Otherwise, if α is a random value, $\mathcal{A}$ was playing game G_ι.

Let G'_0 and G'_1 be the *CKS-light* games played by $\mathcal{B}$ when $b = 0$ and $b = 1$, respectively. Let F denote the event that $\mathcal{B}$ guessed the identities involved in the $(\iota + 1)$-st `test` query correctly (this

would mean that the identities are ID_I and ID_J). Now if F happens and, $\tilde{q}_{R_I} = q_{R_I}$ and $\tilde{q}_{R_J} = q_{R_J}$, then $\mathcal{B}$ simulates the *m-CKS-heavy*$^{++}$ game correctly. Now we assess $\mathcal{B}$'s success probability. It is easy to see that $\Pr[F] = \binom{q_H}{2} \geq 2/q_H^2$ and that $\Pr[\tilde{q}_{R_I} = q_{R_I} \wedge \tilde{q}_{R_J} = q_{R_J}] \geq 1/(q_R+1)^2$. We have:

$$\Pr[\mathcal{B}(G'_0) = 1] = \frac{1}{q_T} \Pr[F] \Pr[\tilde{q}_{R_I} = q_{R_I} \wedge \tilde{q}_{R_J} = q_{R_J}] \sum_{\iota=0}^{q_T-1} \Pr[\mathcal{A}(G_{\iota+1}) = 1]$$

and

$$\Pr[\mathcal{B}(G'_1) = 1] = \frac{1}{q_T} \Pr[F] \Pr[\tilde{q}_{R_I} = q_{R_I} \wedge \tilde{q}_{R_J} = q_{R_J}] \sum_{\iota=0}^{q_T-1} \Pr[\mathcal{A}(G_{\iota}) = 1].$$

Therefore it follows that

$$\begin{aligned}
\mathrm{Adv}_{\mathcal{B}}^{\textit{CKS-light}}(k, q_C, q'_{CR}) &= \left|\Pr[\mathcal{B}(G'_0) = 1] - \Pr[\mathcal{B}(G'_1) = 1]\right| \\
&\geq \frac{2}{q_T q_H^2 (q_R+1)^2} \left| \sum_{\iota=0}^{q_T-1} \Pr[\mathcal{A}(G_{\iota+1}) = 1] - \sum_{\iota=0}^{q_T-1} \Pr[\mathcal{A}(G_{\iota}) = 1] \right| \\
&= \frac{2}{q_T q_H^2 (q_R+1)^2} \left|\Pr[\mathcal{A}(G_0) = 1] - \Pr[\mathcal{A}(G_{q_T}) = 1]\right| \\
&= \frac{2}{q_T q_H^2 (q_R+1)^2} \mathrm{Adv}_{\mathcal{A}}^{\textit{m-CKS-heavy}^{++}}(k, q_H, q_C, q_E, q_{HR}, q_{CR}, q_T, q_R).
\end{aligned}$$

This concludes our proof.

□

Let *CKS-light*$^{++}$, *CKS*$^{++}$, *CKS-heavy*$^{++}$ and *m-CKS-heavy*$^{++}$ be the augmented versions of the *CKS-light*, *CKS*, *CKS-heavy* and *m-CKS-heavy* security models from [18], including re-registration of honest users and corrupt registration of previously registered honest users. Additionally, let *CKS-light*$^{+}$, *CKS*$^{+}$, *CKS-heavy*$^{+}$ and *m-CKS-heavy*$^{+}$ be the augmented versions of the latter models, including only re-registration of honest users.

Theorem B.3 (Polynomial equivalence of all models)**.** *The CKS-light, CKS, CKS-heavy, m-CKS-heavy, CKS-light*$^{+}$*, CKS*$^{+}$*, CKS-heavy*$^{+}$*, m-CKS-heavy*$^{+}$*, CKS-light*$^{++}$*, CKS*$^{++}$*, CKS-heavy*$^{++}$ *and m-CKS-heavy*$^{++}$ *security models are all polynomially equivalent.*

Proof. The proof follows from Theorems B.1 and B.2. Note that because the *CKS-light*$^{++}$, *CKS*$^{++}$ and *CKS-heavy*$^{++}$ security models are limited cases of the *m-CKS-heavy*$^{++}$ model, the non-trivial direction (*m-CKS-heavy*$^{++}$ $\Leftarrow$ *CKS-light*) of Theorem B.2 implies that *CKS-light* security also implies *CKS-light*$^{++}$, *CKS*$^{++}$ and *CKS-heavy*$^{++}$ security. Now, because the *CKS-light*$^{++}$, *CKS*$^{++}$ and *CKS-heavy*$^{++}$ security models are augmented versions of the *CKS-light*, *CKS* and *CKS-heavy* models, respectively, Theorem B.1 also implies that security in the sense of the *CKS-light*$^{++}$, *CKS*$^{++}$ or *CKS-heavy*$^{++}$ model imply security in the *CKS-light* model. Note that the same argument is valid for the *CKS-light*$^{+}$, *CKS*$^{+}$, *CKS-heavy*$^{+}$ and *m-CKS-heavy*$^{+}$. Thus, we see that all security models *including or not* honest re-registration and/or corrupt registration of previously registered honest users are polynomially equivalent. □

A strictly weaker notion: *weakCKS*$^{++}$ security

In order to prove security of our NIKE' scheme from Definition 4.4, we also need to make use of a strictly weaker security notion for NIKE, which we call *weakCKS*$^{++}$ security. This notion requires that, instead of having to distinguish between real and random keys like in all the above mentioned security notions for NIKE, an adversary against a NIKE scheme has to output a current shared key between two honest users. The *weakCKS*$^{++}$ security model is a limited and weaker variant of our *m-CKS-heavy*$^{++}$ model. There are no `test` queries because this is not an indistinguishability based notion. Furthermore, in this security model, an adversary is only allowed to make the following types of queries: $(\texttt{register honest user}, \mathsf{ID})$, $(\texttt{register corrupt user}, \mathsf{ID}, \mathsf{pk})$, $(\texttt{extract}, \mathsf{ID})$ and $(\texttt{corrupt reveal}, \mathsf{ID}_i, \mathsf{ID}_j)$, including the above mentioned re-registration of users when we use the $^{++}$ notation.

The advantage of an adversary $\mathcal{A}$ in the *weakCKS*$^{++}$ game is defined as

$$\mathrm{Adv}_{\mathcal{A}}^{weakCKS^{++}}(k, q_H, q_C, q_E, q_{CR}, q_R) = \Pr[(\mathsf{ID}_i, \mathsf{ID}_j, \texttt{NIKE.SharedKey}(\mathsf{ID}_i, \mathsf{pk}_i, \mathsf{ID}_j, \mathsf{sk}_j)) \leftarrow \mathcal{A}],$$

where q_H, q_C, q_E, q_{CR} and q_R denote the same as in the definition of the *m-CKS-heavy*$^{++}$ model. $\{\mathsf{ID}_i, \mathsf{ID}_j\}$ is any pair of honest users subject to the constraint that $\mathcal{A}$ issued no `extract` queries for ID_i or ID_j after the last honest re-registration of ID_i or ID_j respectively (if it exists) before $\mathcal{A}$ outputs its guess. $\texttt{NIKE.SharedKey}(\mathsf{ID}_i, \mathsf{pk}_i, \mathsf{ID}_j, \mathsf{sk}_j)$ is the current shared key between ID_i and ID_j.

We note that using a similar line of argument as seen in the proof of Theorem B.2, it can be shown that the *weakCKS*$^{++}$ security model is polynomially equivalent to its version without the extra allowance of the above mentioned re-registration of users, which we denote by the *weakCKS* security model. Thus, in the construction of a NIKE' scheme from Definition 4.4, we actually only require a NIKE scheme NIKE which is secure in the sense of the *weakCKS* security notion.

C The UC model of protocol execution

We use the Universal Composability (UC) framework for multi-party protocols defined in [4]. In fact, we employ the variant "GNUC" of UC [23], which differs from the original UC framework in a number of low-level details. (However, both the results in our main part and the high-level description given here are largely independent of such low-level details.)

The real model, static and adaptive attacks. First of all, *parties* (usually denoted by P_1 through P_n) are modeled as *interactive machines (IMs)* and are supposed to run some (fixed) protocol π. There also is an *adversary* (denoted $\mathcal{A}$ and modeled as an IM as well) carrying out attacks on protocol π. Therefore, $\mathcal{A}$ may corrupt parties (in which case it learns the party's current state and the contents of all its tapes, and controls its future actions), and intercept or, when assuming unauthenticated message transfer, also inject messages sent between parties. If $\mathcal{A}$ corrupts parties only *before* the actual protocol run of π takes place (i.e., in its first activations), $\mathcal{A}$ is called *static*, otherwise $\mathcal{A}$ is said to be *adaptive*. The respective local inputs for protocol π are supplied by an *environment machine* (modeled as an IM and denoted $\mathcal{Z}$), which may also read all outputs locally made by the parties and communicate with the adversary. We assume that all involved entities are polynomial-time in the security parameter k. (The notion of polynomial-time is nontrivial in the UC setting, and we refer the reader to [23] for details.)

The ideal model. The model we have just described is called the *real* model of computation. In contrast to this, the *ideal* model of computation is defined just like the real model, with the following exceptions: we have an additional IM called the *ideal functionality* $\mathcal{F}$ that is able to send messages to and receive messages from the parties privately (i.e., without the adversary being able to even intercept these messages). The ideal functionality may not be corrupted by the adversary, yet may send messages to and receive messages from it. Furthermore, the parties $P_1, \ldots, P_n$ are replaced by *dummy parties* $\tilde{P}_1, \ldots, \tilde{P}_n$ which simply forward their respective inputs to $\mathcal{F}$ and take messages received from $\mathcal{F}$ as output. Finally, the adversary in the ideal model is called the *simulator* and denoted $\mathcal{S}$. The only means of attack the simulator has in the ideal model are those of corrupting parties (which has the same effect as in the real model), delaying or even suppressing messages sent from $\mathcal{F}$ to a party, and all actions that are explicitly specified in $\mathcal{F}$. (We note, however, that $\mathcal{F}_{NIKE}$ offers no possibility for $\mathcal{S}$ to delay or even suppress output.)

$\mathcal{S}$ has no access to the contents of the messages sent from $\mathcal{F}$ to the dummy parties (except in the case that the receiving party is corrupted) nor are there any messages actually sent between (uncorrupted) parties $\mathcal{S}$ could intercept. Intuitively, the ideal model of computation (or, more precisely, the ideal functionality $\mathcal{F}$ itself) should represent what we ideally expect a protocol to do. In fact, for a number of standard tasks, there are formulations as such ideal functionalities (see, e.g., [4]).

The security definition. To decide whether or not a given protocol π does what we would ideally expect some ideal functionality $\mathcal{F}$ to do, the UC and GNUC frameworks use a *simulation*-based approach: at a time of its choice, $\mathcal{Z}$ may halt and leave a bit of output on its output tape. The random variable describing $\mathcal{Z}$'s output will be denoted by $\text{Exec}[\pi, \mathcal{A}, \mathcal{Z}]$, resp. $\text{Exec}[\mathcal{F}, \mathcal{S}, \mathcal{Z}]$. Now if for any adversary $\mathcal{A}$ in the real model, there exists a simulator $\mathcal{S}$ in the ideal model such that for *any* environment $\mathcal{Z}$, we have that

$$|\Pr[\text{Exec}[\pi, \mathcal{A}, \mathcal{Z}] = 1] - \Pr[\text{Exec}[\mathcal{F}, \mathcal{S}, \mathcal{Z}] = 1]|$$

is negligible as a function in the security parameter, then protocol π is said to *securely realize* functionality $\mathcal{F}$. Intuitively, this means that any attack carried out by adversary $\mathcal{A}$ in the real model can also be carried out in the idealized modeling with an ideal functionality by the simulator $\mathcal{S}$ (hence the name), such that no environment is able to tell the difference.

The hybrid model and universal composition. To allow for a modular protocol design, there also exists the *$\mathcal{F}$-hybrid model of computation* (for an arbitrary ideal functionality $\mathcal{F}$). Briefly, this model is identical to the real model of computation, but the parties have access to an unbounded number of instances of $\mathcal{F}$. The modularity of the hybrid model is legitimated by the fundamental *composition theorem* of [4]. To describe the theorem statement, suppose that protocol ϕ securely realizes functionality $\mathcal{F}$. Then, in any protocol π running in the $\mathcal{F}$-hybrid model, all $\mathcal{F}$-invocations can be substituted by invocations of ϕ without losing security. Specifically, for every real-life adversary $\mathcal{A}$, there is a hybrid-model adversary $\mathcal{H}$ such that no environment can tell whether it is interacting with $\mathcal{A}$ and π (with $\mathcal{F}$-instances substituted by invocations of ϕ) in the real model, or with $\mathcal{H}$ and π in the $\mathcal{F}$-hybrid model.

D The ideal functionality $\mathcal{F}_{RO}$

We present here a version of the random oracle ideal functionality $\mathcal{F}_{RO}$. For simplicity, since we need only one instance of $\mathcal{F}_{RO}$, we omit the use of session identifiers and stress that in our

$\mathcal{F}_{RO}$-hybrid model, all parties always call the same instance of $\mathcal{F}_{RO}$.

> $\mathcal{F}_{RO}$ proceeds as follows, running on security parameter k, with parties $\mathsf{P}_1, \ldots, \mathsf{P}_n$ and an adversary.
>
> - $\mathcal{F}_{RO}$ keeps a list L (which is initially empty) of pairs of bitstrings.
> - Upon receiving a value m (with $m \in \{0,1\}^*$) from some party P_i or from the adversary, do:
>   - If there is a pair $(m, \tilde{h})$ for some $\tilde{h} \in \{0,1\}^k$ in the list L, set $h := \tilde{h}$.
>   - If there is no such pair, choose uniformly $h \in \{0,1\}^k$ and store the pair (m, h) in L.
>
>   Once h is set, reply to the activating machine (*i.e.*, either P_i or the adversary) with h.

Functionality $\mathcal{F}_{RO}$.

E The ideal functionality $\mathcal{F}_{CRS}$

Throughout the paper we make use of a hybrid functionality $\mathcal{F}_{CRS}$, which provides all parties and the adversary with a common reference string (CRS). The CRS is drawn from a specific distribution, in our case, the output distribution of the algorithm $\texttt{NIKE.CommonSetup}(1^k)$. The functionality $\mathcal{F}_{CRS}$ is described below.

> $\mathcal{F}_{CRS}$ proceeds as follows, running on security parameter k, with parties $\mathsf{P}_1, \ldots, \mathsf{P}_n$ and an adversary.
>
> - Upon receiving the first input of the form (`parameters`), compute $params \xleftarrow{\$} \texttt{NIKE.CommonSetup}(1^k)$ and send $params$ to the activating entity. From now on, always answer (`parameters`) requests with $params$.

Functionality $\mathcal{F}_{CRS}$.

F Proof of Theorem 4.2

Proof. Let $\mathcal{B}$ be a *CKS-light* adversary. Out of $\mathcal{B}$, we construct two environments, $\mathcal{Z}$ and $\tilde{\mathcal{Z}}$, where $\mathcal{Z}$'s view in a real execution of `NIKE` corresponds to an instance of a *CKS-light* game with $\mathcal{B}$ and challenge bit $b = 0$ (which means that `test` queries are answered with real keys in the *CKS-light* game). $\tilde{\mathcal{Z}}$ transforms $\mathcal{Z}$'s view to an instance of a *CKS-light* game with $b = 1$.

We start with a description of $\mathcal{Z}$. $\mathcal{Z}$ internally simulates $\mathcal{B}$ and translates $\mathcal{B}$'s queries in the *CKS-light* security game into requests to honest `NIKE` parties, resp. to the adversary. This is done using an adaptation of the query translations from Table 1; for instance, when $\mathcal{B}$ makes a $(\texttt{register corrupt user}, \mathsf{P}_i, \mathsf{pk}_i)$ request, then $\mathcal{Z}$ asks the adversary to corrupt P_i and to let P_i register pk_i as its public key.[10] In the *CKS-light* security model there are no re-registrations of honest users, so there are no $(\texttt{renew}, \mathsf{P}_i)$ requests in the UC world. $(\texttt{register honest user}, \mathsf{P}_i)$

[10] We stress that this still corresponds to a *static* corruption in the UC model (cf. [6, Section 6.7]): since $\mathcal{B}$ is a *CKS-light* adversary, party P_i has not been invoked before; thus, $\mathcal{Z}$ only requests corruptions of freshly created parties.

queries for the *two* honest parties, say P_1 and P_2, are handled by issuing $(\texttt{register}, \mathsf{P}_i)$ $(i \in \{1,2\})$ queries to P_i and P_j. When $\mathcal{B}$ outputs a decision bit b', $\mathcal{Z}$ also outputs b'.

It is easy to see that a real execution of NIKE with such a $\mathcal{Z}$ corresponds to a *CKS-light* game with $b = 0$ (recall that $b = 0$ means that the *CKS-light* challenger $\mathcal{C}$ will answer test queries by running NIKE.SharedKey). Hence, $\Pr[\text{Exec}[\texttt{NIKE}, \mathcal{A}, \mathcal{Z}] = 1] = \Pr[1 \leftarrow \mathcal{B} \mid b = 0]$.

Next we define the environment $\tilde{\mathcal{Z}}$ as a modification of $\mathcal{Z}$. $\tilde{\mathcal{Z}}$ internally simulates $\mathcal{Z}$. However, when $\mathcal{Z}$ initializes a session including the only two honest parties, $\tilde{\mathcal{Z}}$ answers with a randomly chosen key instead of forwarding the initialization and using the output of the UC party as answer to $\mathcal{Z}$. Note that in the ideal world, where keys for honest sessions are randomly chosen anyway (by $\mathcal{F}_{NIKE}$), this does not alter $\mathcal{Z}$'s view. In particular, if we let $\tilde{\mathcal{Z}}$ adopt $\mathcal{Z}$'s output bit, we have $\text{Exec}[\mathcal{F}_{NIKE}, \mathcal{S}, \mathcal{Z}] = \text{Exec}[\mathcal{F}_{NIKE}, \mathcal{S}, \tilde{\mathcal{Z}}]$ (for any $\mathcal{S}$). Again it is easy to see that in a real execution of NIKE, $\tilde{\mathcal{Z}}$ simulates a *CKS-light* game for $\mathcal{Z}$ with $b = 1$. Hence, $\Pr[\text{Exec}[\texttt{NIKE}, \mathcal{A}, \tilde{\mathcal{Z}}] = 1] = \Pr[1 \leftarrow \mathcal{B} \mid b = 1]$.

Finally, as NIKE is UC-secure w.r.t. static environments, the distinguishing advantages of both $\mathcal{Z}$ and $\tilde{\mathcal{Z}}$ must be negligible. Putting things together, we get

$$\begin{aligned}
\Pr[1 \leftarrow \mathcal{B} | b = 0] &= \Pr[\text{Exec}[\texttt{NIKE}, \mathcal{A}, \mathcal{Z}] = 1] \\
&\approx \Pr[\text{Exec}[\mathcal{F}_{NIKE}, \mathcal{S}, \mathcal{Z}] = 1] \\
&= \Pr[\text{Exec}[\mathcal{F}_{NIKE}, \mathcal{S}, \tilde{\mathcal{Z}}] = 1] \\
&\approx \Pr[\text{Exec}[\texttt{NIKE}, \mathcal{A}, \tilde{\mathcal{Z}}] = 1] \\
&= \Pr[1 \leftarrow \mathcal{B} | b = 1].
\end{aligned}$$

Therefore, NIKE is *CKS-light*-secure. □

Why we use two different environments in the proof of Theorem 4.2. One might be tempted to directly associate the ideal UC setting with $\mathcal{F}_{NIKE}$ with the *CKS-light* game with $b = 1$ (i.e., with random keys). This would seem to enable a simpler proof of Theorem 4.2. However, note that an ideal world simulator $\mathcal{S}$ controls $\mathcal{F}_{CRS}$ and $\mathcal{F}_{KR}^1$. Hence, $\mathcal{S}$ is able to generate parameters and public keys of honest users in a malicious way. This does not correspond to a *CKS-light* game anymore, where the parameters and public keys of honest users are honestly chosen by the *CKS-light* challenger.

G Proof of Theorem 4.5

Proof. Let $\mathcal{A}$ be the dummy adversary in the UC setting. We construct a simulator $\mathcal{S}$ that interacts with $\mathcal{Z}$ and $\mathcal{F}_{NIKE}$ such that for all distinguishing environments $\mathcal{Z}$,

$$\text{Exec}[\mathcal{F}_{NIKE}, \mathcal{S}, \mathcal{Z}] \approx \text{Exec}[\texttt{NIKE'}, \mathcal{A}, \mathcal{Z}].$$

We proceed with a description of $\mathcal{S}$. $\mathcal{S}$ internally simulates the real life execution of NIKE' with $\mathcal{A}$. $\mathcal{S}$ maintains a list of corrupted parties and a list Λ with entries of the form $(\mathsf{P}_i, \mathsf{pk}_i, \mathsf{sk}_i)$, containing party identifiers and their public key/secret key pairs. $\mathcal{S}$ also maintains a list Λ_H to handle random oracle queries. We now give a definition of $\mathcal{S}$ by specifying its reaction to invocations from any entity in the ideal world, including the internally simulated real life execution:

(parameters) **from any entity.** $\mathcal{S}$ runs $params \xleftarrow{\$} \texttt{NIKE.CommonSetup}(1^k)$ once and always answers this request with $params$.

$(H, \mathsf{P}_i, \mathsf{P}_j, h)$ **from any entity.** $\mathcal{S}$ maintains a list Λ_H (initially empty) in order to handle Random Oracle (RO) queries. Upon receiving a tuple $(\mathsf{P}_i, \mathsf{P}_j, h)$, w.l.o.g. $\mathsf{P}_i, \mathsf{P}_j \in \{0,1\}^*$, $\mathsf{P}_i < \mathsf{P}_j$, and $h \in \{0,1\}^k$ (the shared key space of NIKE), $\mathcal{S}$ answers RO queries as follows: $\mathcal{S}$ checks if there already exists an entry $((\mathsf{P}_i, \mathsf{P}_j, h), r)$ in the list Λ_H. If so, $\mathcal{S}$ returns r. Otherwise, $\mathcal{S}$ uniformly chooses $r' \xleftarrow{\$} \{0,1\}^{k'}$, records $((\mathsf{P}_i, \mathsf{P}_j, h), r')$ in Λ_H and returns r'. $\mathcal{S}$ returns $\perp$ if $h \notin \{0,1\}^k$ (which is represented by $h = \perp$).

$(\texttt{register}, \mathsf{P}_i)$ **from** $\mathcal{F}_{NIKE}$**.** $\mathcal{S}$ checks if there exists an entry of the form $(\mathsf{P}_i, \cdot, \cdot)$ in Λ. If not, $\mathcal{S}$ runs $\texttt{NIKE.KeyGen}(\mathit{params}, \mathsf{P}_i)$ to obtain a valid public key/secret key pair $(\mathsf{pk}_i, \mathsf{sk}_i)$ for P_i and stores $(\mathsf{P}_i, \mathsf{pk}_i, \mathsf{sk}_i)$ in Λ (W.l.o.g. we assume that $\mathcal{S}$ has already computed *params*). $\mathcal{S}$ then sends $(\mathsf{P}_i, \texttt{registered})$ to $\mathcal{F}_{NIKE}$, waits until $\mathcal{F}_{NIKE}$ sends back $(\mathsf{P}_i, \texttt{registered})$, then sends $(\texttt{register}, \mathsf{P}_i)$ to $\mathcal{Z}$. Note that $\mathcal{F}_{NIKE}$ keeps sk_i private.

$(\texttt{init}, \mathsf{P}_i, \mathsf{P}_j)$ **from** $\mathcal{F}_{NIKE}$**.** The receipt of this input implies that $\mathsf{P}_i, \mathsf{P}_j$ are not both honest. We can assume P_j is corrupted, because $\mathcal{S}$ would not send $(\texttt{init}, \mathsf{P}_i, \mathsf{P}_j)$ through a corrupted P_i. $\mathcal{S}$ computes $\texttt{NIKE.SharedKey}(\mathsf{P}_j, \mathsf{pk}_j, \mathsf{P}_i, \mathsf{sk}_i)$, sets

$$K_{i,j} = \begin{cases} H(\mathsf{P}_i, \mathsf{P}_j, \texttt{NIKE.SharedKey}(\mathsf{P}_j, \mathsf{pk}_j, \mathsf{P}_i, \mathsf{sk}_i)), & \text{if } \mathsf{P}_i < \mathsf{P}_j \text{ or} \\ H(\mathsf{P}_j, \mathsf{P}_i, \texttt{NIKE.SharedKey}(\mathsf{P}_j, \mathsf{pk}_j, \mathsf{P}_i, \mathsf{sk}_i)), & \text{if } \mathsf{P}_j < \mathsf{P}_i, \end{cases}$$

and sends $(\mathsf{P}_i, \mathsf{P}_j, K_{i,j})$ to $\mathcal{F}_{NIKE}$.

$(\texttt{renew}, \mathsf{P}_i)$ **from** $\mathcal{F}_{NIKE}$**.** $\mathcal{S}$ obtains $(\mathsf{pk}_i, \mathsf{sk}_i) \xleftarrow{\$} \texttt{NIKE.KeyGen}(\mathit{params}, \mathsf{P}_i)$ and stores $(\mathsf{P}_i, \mathsf{pk}_i, \mathsf{sk}_i)$ in Λ, overwriting any existing entry for P_i. $\mathcal{S}$ then sends $(\mathsf{P}_i, \texttt{registered})$ to $\mathcal{F}_{NIKE}$, waits until $\mathcal{F}_{NIKE}$ sends back $(\mathsf{P}_i, \texttt{registered})$ and then sends $(\texttt{register}, \mathsf{P}_i)$ to $\mathcal{Z}$.

$(\texttt{corrupt}, \mathsf{P}_i)$ **from** $\mathcal{Z}$**.** If P_i has no public key registered, $\mathcal{Z}$ expects to learn an empty internal state. Otherwise, if there exists an entry $(\mathsf{P}_i, \mathsf{pk}_i, \mathsf{sk}_i)$ in Λ, for the honest party P_i, $\mathcal{S}$ sends sk_i to $\mathcal{Z}$. $\mathcal{S}$ corrupts P_i and learns shared key values $(\mathsf{P}_i, \mathsf{P}_j, r_{ij})$[11]. For every tuple $\{\mathsf{P}_i, \mathsf{P}_j\}$ $(\mathsf{P}_j \in \Lambda)$, $\mathcal{S}$ computes $h = \texttt{NIKE.SharedKey}(\mathsf{P}_j, \mathsf{pk}_j, \mathsf{P}_i, \mathsf{sk}_i)$ and records $((\mathsf{P}_i, \mathsf{P}_j, h), r_{ij})$ in Λ_H if $\mathsf{P}_i < \mathsf{P}_j$, or $((\mathsf{P}_j, \mathsf{P}_i, h), r_{ij})$ if $\mathsf{P}_j < \mathsf{P}_i$. $\mathcal{S}$ aborts the simulation if there already exists an entry $((\{\mathsf{P}_i, \mathsf{P}_j\}, h), \cdot)$ in the list Λ_H.

$(\texttt{register}, \mathsf{P}_i, \mathsf{pk}_i)$ **from** $\mathcal{Z}$**.** We can assume that P_i is corrupted. $\mathcal{S}$ adds $(\mathsf{P}_i, \mathsf{pk}_i, \perp)$ to Λ, overwriting any existing entry for P_i. $\mathcal{S}$ then sends $(\mathsf{P}_i, \texttt{registered})$ to $\mathcal{F}_{NIKE}$, waits until $\mathcal{F}_{NIKE}$ sends back $(\mathsf{P}_i, \texttt{registered})$ and sends $(\texttt{register}, \mathsf{P}_i)$ to $\mathcal{Z}$.

$(\texttt{lookup}, \mathsf{P}_i)$ **from any entity.** If there is an entry $(\mathsf{P}_i, \mathsf{pk}_i, \cdot)$ in Λ, return $(\mathsf{P}_i, \mathsf{pk}_i)$. Otherwise, return $\perp$.

We will show that this simulation is in fact *perfect*, unless a certain event BAD occurs. Concretely, let BAD denote the event that at any time, $\mathcal{Z}$ queries $(H, \mathsf{P}_i, \mathsf{P}_j, h)$ for two honest parties $\mathsf{P}_i, \mathsf{P}_j$ that have both registered their public keys, and $h = \texttt{NIKE.SharedKey}(\mathsf{P}_i, \mathsf{pk}_i, \mathsf{P}_j, \mathsf{sk}_j)$. (Note that since P_i and P_j are both honest, their secret keys sk_i and sk_j are well-defined.) Observe that BAD is necessary for $\mathcal{S}$ to abort. In principle, BAD can be defined both in the real and the ideal UC game; however, when we write BAD, we mean BAD in the ideal UC game.

Claim 1: $|\Pr[\text{Exec}[\texttt{NIKE'}, \mathcal{A}, \mathcal{Z}] = 1] - \Pr[\text{Exec}[\mathcal{F}_{NIKE}, \mathcal{S}, \mathcal{Z}] = 1]| \leq \Pr[\texttt{BAD}]$.

[11] Here we assume non-erasing parties, as it is done in [4], meaning that the internal state of a dummy party contains everything the party has relayed so far. We note that GNUC ([23]) does not specify whether dummy parties are erasing. However, our simulation can be slightly modified to work in a model with erasing parties. The modified simulator, controlling corrupted parties, will simply request all possible shared keys (with other registered parties) from $\mathcal{F}_{NIKE}$ through the corrupted parties again and program the random oracle with these keys. This can be done in a time polynomially bounded in the number of inputs $\mathcal{Z}$ gave so far (i.e., in the number of invoked parties).

Proof. In this proof we define a sequence of indistinguishable games G_0, G_1, G_2, G_3 and G_4, all defined over the same probability space. Starting with G_0, the real execution of the protocol NIKE', we make slight modifications between successive games, in such a way that the view of an environment $\mathcal{Z}$ is indistinguishable among the games. The last game, G_4, will be indistinguishable from an ideal execution with $\mathcal{S}$ as defined above, when conditioned on $\neg$BAD. In G_1, we basically regroup machines and add some relays. We explain how $\mathcal{S}$ internally simulates the parties and the functionalities $\mathcal{F}_{RO}$, $\mathcal{F}_{CRS}$ and $\mathcal{F}_{KR}^1$. In G_2 and G_3, we let one of the relays store information and make some decisions. Finally, in G_4, we make some modifications, transforming one of the added relays into the ideal key exchange functionality $\mathcal{F}_{NIKE}$.

Game G_0: Real protocol run. This is the real protocol run (involving all the parties in the NIKE' protocol and an adversary $\mathcal{A}$, controlling some of the parties) with the ideal functionalities $\mathcal{F}_{CRS}$, $\mathcal{F}_{KR}^1$ and $\mathcal{F}_{RO}$. The situation is similar to the left-hand side of Figure 1, but with an extra ideal functionality $\mathcal{F}_{RO}$.

Game G_1: Regrouping of machines and addition of relays. We modify the network of G_0 by regrouping all the machines but $\mathcal{Z}$ into one machine and call it $\mathcal{S}$. $\mathcal{S}$ simulates all parties in the protocol, as well as the helping functionalities. Then, we add to the network single relays representing each party in the NIKE' protocol, and another relay, which we call $\mathcal{F}$ (see right-hand side of Figure 1). For now, $\mathcal{F}$ basically only relays the connections between the relays representing each party in the protocol, and the parties themselves (internally simulated by $\mathcal{S}$).

We now explain how $\mathcal{S}$ simulates the helping functionalities.

The functionality $\mathcal{F}_{CRS}$ is implemented by $\mathcal{S}$ by obtaining $params \xleftarrow{\$} \texttt{NIKE.CommonSetup}(1^k)$ once. $\mathcal{S}$ always answers queries to $\mathcal{F}_{CRS}$ with $params$.

In order to properly manage queries for the random oracle H, which will be made by the simulated parties or by the environment $\mathcal{Z}$, $\mathcal{S}$ maintais a list Λ_H. $\mathcal{S}$ updates the list according to the above description of how our simulator $\mathcal{S}$ handles RO queries.

For the key registration functionality, we allow $\mathcal{S}$ to maintain a list Λ with entries of the form $(\mathsf{P}_i, \mathsf{pk}_i, \mathsf{sk}_i)$, containing a party identifier, its public key and its secret key (if defined). Whenever a party P_i (internally simulated by $\mathcal{S}$) runs $\texttt{NIKE.KeyGen}(params, \mathsf{P}_i)$ to obtain a valid public key/secret key pair $(\mathsf{pk}_i, \mathsf{sk}_i)$, $\mathcal{S}$ stores $(\mathsf{P}_i, \mathsf{pk}_i, \mathsf{sk}_i)$ in Λ, overwriting any existing entry for P_i, and sends $(\texttt{register}, \mathsf{P}_i)$ to $\mathcal{Z}$. Moreover, whenever the environment $\mathcal{Z}$ tries to register a party with the ideal functionality $\mathcal{F}_{KR}^1$ by sending $(\texttt{register}, \mathsf{P}_i, \mathsf{pk}_i)$, $\mathcal{S}$ sends $(\texttt{register}, \mathsf{P}_i)$ to $\mathcal{Z}$, and adds $(\mathsf{P}_i, \mathsf{pk}_i, \perp)$ to Λ, overwriting any existing entry for P_i. Additionally, $\mathcal{S}$ can easily handle $(\texttt{lookup}, \mathsf{P}_i)$ by searching P_i in Λ.

Note that because $\mathcal{S}$ simulates all parties that $\mathcal{Z}$ might invoke during the protocol run, $\mathcal{S}$ has access to all parties' internal states. $\mathcal{S}$ is thus able to answer all the corrupt queries made by $\mathcal{Z}$ or even to forward public keys to $\mathcal{Z}$.

As this new network is basically only a regrouping of the previous network and because $\mathcal{S}$ perfectly simulates the parties and the helping functionalities, it holds that $\mathcal{Z}$'s view is still the same as in game G_1.

Game G_2: Allowing $\mathcal{F}$ to store information and make decisions. This game is exactly like Game G_3 in proof of Theorem 4.1. We let $\mathcal{S}$ send $(\mathsf{P}_i, \texttt{registered})$ to $\mathcal{F}$ whenever a party successfully registers a public key pk_i with $\mathcal{F}_{KR}^1$. $\mathcal{F}$ answers such a message by bouncing it, i.e. sending $(\mathsf{P}_i, \texttt{registered})$ to $\mathcal{S}$. We also let $\mathcal{F}$ store information about the registered parties and make decisions regarding whether to forward tuples, containing shared keys, coming from $\mathcal{S}$, to

parties. Additionally, we assume $\mathcal{F}$ knows which parties are corrupted. $\mathcal{F}$ maintains a list Λ_{reg}. On input $(\mathsf{P}_i, \texttt{registered})$ from $\mathcal{S}$, if $\mathsf{P}_i \notin \Lambda_{reg}$, $\mathcal{F}$ adds P_i to Λ_{reg}. Upon receiving $(\texttt{init}, \mathsf{P}_i, \mathsf{P}_j)$, if $\mathsf{P}_j \notin \Lambda_{reg}$, $\mathcal{F}$ sends $(\mathsf{P}_i, \mathsf{P}_j, \perp)$ to P_i. Else $\mathcal{F}$ relays $(\texttt{init}, \mathsf{P}_i, \mathsf{P}_j)$ to $\mathcal{S}$ and receives an answer $(\mathsf{P}_i, \mathsf{P}_j, K_{i,j})$. $\mathcal{F}$ relays $(\mathsf{P}_i, \mathsf{P}_j, K_{i,j})$ to P_i.

It is easy to see that the output of a party P_i in G_1 is $(\mathsf{P}_i, \mathsf{P}_j, \perp)$ if and only if the output of a party in G_2 is $(\mathsf{P}_i, \mathsf{P}_j, \perp)$. Thus $\mathcal{Z}$'s view in G_2 is still the same as in G_1.

Game G_3: More lists and more decisions for $\mathcal{F}$. Here we let $\mathcal{F}$ maintain two other lists, Λ_{keys} and Λ_{renew}. Before relaying any tuple $(\mathsf{P}_i, \mathsf{P}_j, key)$, where $key \neq \perp$, to a party P_i, $\mathcal{F}$ adds $(\{\mathsf{P}_i, \mathsf{P}_j\}, key)$ to Λ_{keys}. Upon receipt of $(\texttt{renew}, \mathsf{P}_i)$ from P_i, $\mathcal{F}$ stores P_i in Λ_{renew}. Upon receipt of $(\mathsf{P}_i, \texttt{registered})$ from $\mathcal{S}$, if $\mathsf{P}_i \in \Lambda_{renew}$, $\mathcal{F}$ deletes every existing entry $(\{\mathsf{P}_i, \cdot\}, key)$ from Λ_{keys} and deletes P_i from Λ_{renew}. Upon receipt of $(\texttt{init}, \mathsf{P}_i, \mathsf{P}_j)$ with $\mathsf{P}_j \in \Lambda_{reg}$ we let $\mathcal{F}$ check the list Λ_{keys} for an entry $(\{\mathsf{P}_i, \mathsf{P}_j\}, key)$. If there is one, $\mathcal{F}$ does not relay $(\texttt{init}, \mathsf{P}_i, \mathsf{P}_j)$ to the adversary and instead returns $(\{\mathsf{P}_i, \mathsf{P}_j\}, key)$ to P_i right away.

The output of $\mathcal{F}$ is the same as in Game G_2, because any entry in Λ_{keys} was computed by $\mathcal{S}$ and therefore matches the answer of $\mathcal{S}$ to the `init` request in Game G_2.

Game G_4: Building the ideal functionality $\mathcal{F}_{NIKE}$. Upon receipt of an $(\texttt{init}, \mathsf{P}_i, \mathsf{P}_j)$ request, where P_i and P_j are honest and $\mathsf{P}_j \in \Lambda_{keys}$, we prevent $\mathcal{F}$ from relaying $(\texttt{init}, \mathsf{P}_i, \mathsf{P}_j)$ to the adversary. Instead, $\mathcal{F}$ checks if there already exists an entry $(\{\mathsf{P}_i, \mathsf{P}_j\}, key)$ in Λ_{keys}. If not, $\mathcal{F}$ sets $key \xleftarrow{\$} \{0,1\}^{k'}$ (the shared key space of `NIKE'`) and stores $(\{\mathsf{P}_i, \mathsf{P}_j\}, key)$ in Λ_{keys}. $\mathcal{F}$ sends $(\mathsf{P}_i, \mathsf{P}_j, key)$ to P_i. Upon corruption of a party P_i, $\mathcal{S}$ will learn the shared key values $(\mathsf{P}_i, \mathsf{P}_j, r_{ij})$ that were randomly chosen by $\mathcal{F}$. $\mathcal{S}$ then adds $((\mathsf{P}_i, \mathsf{P}_j, \texttt{NIKE.SharedKey}(\mathsf{P}_j, \mathsf{pk}_j, \mathsf{P}_i, \mathsf{sk}_i)), r_{ij})$ (w.l.o.g. assuming that $\mathsf{P}_i < \mathsf{P}_j$) to Λ_H. We stress that in this game, we assume that $\mathcal{Z}$ makes no queries $H(\mathsf{P}_i, \mathsf{P}_j, \texttt{NIKE.SharedKey}(\mathsf{P}_j, \mathsf{pk}_j, \mathsf{P}_i, \mathsf{sk}_i))$ to the random oracle, where P_i and P_j are both honest.

To conclude the proof, we see that in both games G_3 and G_4, the shared key between two honest parties is a randomly generated value from $\{0,1\}^{k'}$, the space of shared keys of `NIKE'`. Thus, games G_3 and G_4 are indistinguishable from $\mathcal{Z}$'s point of view. □

Claim 2: $\Pr[\texttt{BAD}] \leq \text{negl}$.

Proof. Besides running with $\mathcal{Z}$ and $\mathcal{F}_{NIKE}$, we let $\mathcal{S}$ play a *weakCKS*$^{++}$ security game. $\mathcal{S}$ has to be slightly modified to act as a translator between the UC setting and the *weakCKS*$^{++}$ game. We mainly have to specify how $\mathcal{S}$ registers users in the *weakCKS*$^{++}$ game. For the sake of simplicity we only list the modifications of $\mathcal{S}$.

Let $\mathcal{C}$ be $\mathcal{S}$'s challenger in the *weakCKS*$^{++}$ security game.

- Instead of running $\texttt{NIKE.CommonSetup}(1^k)$, $\mathcal{S}$ obtains *params* from $\mathcal{C}$.
- If an honest party P_i needs to register a public key (i.e. after $\mathcal{S}$ received the first $(\texttt{register}, \mathsf{P}_i)$), then, instead of generating a public/secret key pair by running $\texttt{NIKE.KeyGen}(params, \mathsf{P}_i)$, $\mathcal{S}$ registers P_i in the *weakCKS*$^{++}$ game and gets pk_i from $\mathcal{C}$.
- Upon receiving $(\texttt{renew}, \mathsf{P}_i)$ from $\mathcal{F}_{NIKE}$, $\mathcal{S}$ re-registers the honest P_i in the *weakCKS*$^{++}$ game to obtain a new pk_i for this party.
- Upon receiving $(\texttt{register}, \mathsf{P}_i, \mathsf{pk}_i)$ from environment $\mathcal{Z}$, $\mathcal{S}$ simulates the $\mathcal{F}_{KR}^1$ functionality and issues a $(\texttt{register corrupt user}, \mathsf{P}_i, \mathsf{pk}_i)$ query to $\mathcal{C}$.
- If $\mathcal{F}_{NIKE}$ lets $\mathcal{S}$ determine the key of a corrupted session, $\mathcal{S}$ queries $\mathcal{C}$ with a `corrupt reveal` query to obtain a shared key and sends it to $\mathcal{F}_{NIKE}$.

- Upon corruption of a formerly honest party P_i, $\mathcal{S}$ has to inform $\mathcal{Z}$ about the internal state of P_i. This state is empty if P_i so far did not obtain any input from $\mathcal{Z}$, which means that P_i does not exist in the *weakCKS*$^{++}$ game yet. If P_i already obtained an input from $\mathcal{Z}$, $\mathcal{S}$ queries $\mathcal{C}$ for the secret key sk_i by issuing an `extract` query for P_i. Notice that $\mathcal{S}$ will then be able to compute $\texttt{NIKE.SharedKey}(\mathsf{P}_j, \mathsf{pk}_j, \mathsf{P}_i, \mathsf{sk}_i)$ for any registered party P_j. $\mathcal{S}$ sends sk_i to $\mathcal{Z}$.

The output of $\mathcal{S}$ towards other entities is distributed exactly as before because $\mathcal{C}$ uses the same algorithms to generate *params*, public/secret key pairs and shared keys. The modified simulation guarantees that the registered users in the *weakCKS*$^{++}$ game correspond to those parties who registered a public key with $\mathcal{F}_{KR}^{1}$ in the internal simulation of `NIKE'`.

Let q_{RO} denote the total number of RO queries made by $\mathcal{Z}$. Then $\mathcal{S}$ will try to win the *weakCKS*$^{++}$ game by randomly choosing one RO entry and submitting the preimage as guess to $\mathcal{C}$. (The hope is that this guessed RO entry is responsible for `BAD`.) $\mathcal{S}$ picks the right entry with probability $1/q_{RO}$. Thus the advantage of $\mathcal{S}$ in the *weakCKS*$^{++}$ game is

$$Adv_{\mathcal{S}}^{weakCKS^{++}} = \Pr[\texttt{BAD}]/q_{RO}.$$

If $\Pr[\texttt{BAD}]$ is non-negligible, this contradicts the *weakCKS*$^{++}$ security of NIKE. □

The theorem follows from **Claim 1** and **Claim 2**. □