One-Round Key Exchange with Strong Security: An Efficient and Generic Construction in the Standard Model

Florian Bergsma(✉), Tibor Jager, and Jörg Schwenk

Horst Görtz Institute for IT Security, Ruhr-University Bochum, Bochum, Germany
{florian.bergsma,tibor.jager,joerg.schwenk}@rub.de

Abstract. One-round authenticated key exchange (ORKE) is an established research area, with many prominent protocol constructions like HMQV (Krawczyk, CRYPTO 2005) and Naxos (La Macchia *et al.*, ProvSec 2007), and many slightly different, strong security models. Most constructions combine ephemeral and static Diffie-Hellman Key Exchange (DHKE), in a manner often closely tied to the underlying security model.

We give a generic construction of ORKE protocols from general assumptions, with security in the standard model, and in a strong security model where the attacker is even allowed to learn the randomness or the long-term secret of either party in the target session. The only restriction is that the attacker must not learn *both* the randomness *and* the long-term secret of one party of the target session, since this would allow him to recompute all internal states of this party, including the session key.

This is the first such construction that does not rely on random oracles. The construction is intuitive, relatively simple, and efficient. It uses only standard primitives, namely non-interactive key exchange, a digital signature scheme, and a pseudorandom function, with standard security properties, as building blocks.

Keywords: One-round key exchange · eCK security · Provable security

1 Introduction

Key Exchange Protocols and Their Security. Interactive key exchange protocols are fundamental cryptographic building blocks. Two-party protocols, where two parties A and B exchange messages in order to establish a common secret k_{AB}, are particularly important in practice. Popular examples are SSL/TLS [13], SSH [34], and IPSec IKE [22].

Following the seminal works of Bellare and Rogaway (BR) [1] and Canetti and Krawczyk [8], security for such protocols is usually defined with respect to *active* attackers [23,25,32], which may intercept, read, alter, replay, or drop any

J. Katz (Ed.): PKC 2015, LNCS 9020, pp. 477–494, 2015.
DOI: 10.1007/978-3-662-46447-2_21

message transmitted between parties (see Section 3.3 for a precise definition). An attacker in such a security model interacts with a collection of oracles $\pi_1^1, \ldots, \pi_d^\ell$, where all oracles $\pi_i^1, \ldots, \pi_i^d$ share the same long-term public and secret keys of party P_i. An adversary breaks the security of the protocol, if she is able to distinguish the session key k shared between two oracles π_i^s and π_j^t from a random value from the same distribution. To this end, the attacker may ask a $\mathsf{Test}(i, s)$-query to oracle π_i^s. Oracle π_i^s returns either the real key k or a random value, each with probability 1/2.

Typical security models also allow the attacker to *corrupt* selected parties, that is, to learn their long-term secret keys, or to *reveal keys*, that is, to learn the shared keys of sessions which are not related to the Test session. Stronger models [8,23,25,32] allow the attacker furthermore to learn the randomness used by an oracle (which is easy to define clearly), or even internal computation states (which are difficult to define precisely).

One-Round Key Exchange. In this paper we consider *one-round key exchange* (ORKE) protocols, where two parties are able to establish a key in a single round. Such protocols are particularly interesting, due to their simplicity and their efficiency in terms of messages exchanged between parties.

In a (public-key, two-party) ORKE protocol, only two messages are exchanged between two parties A and B. If (pk_A, sk_A) is the public key pair of A, and (pk_B, sk_B) that of B, key establishment proceeds as follows. Party A chooses a random nonce r_A, computes a message $m_A = f(sk_A, pk_B, r_A)$, and sends m_A to B. B chooses a random nonce r_B and responds with message $m_B = f(sk_B, pk_A, r_B)$ (cf. Section 3.2). Note that m_B does not depend on m_A, thus, messages m_A and m_B may be computed and sent *simultaneously* in one round. The key is computed by evaluating a function g with $g(sk_A, pk_B, r_A, m_B) = g(sk_B, pk_A, r_B, m_A)$.

Security Models. Some combinations of adversarial queries lead to trivial attacks, these trivial attacks must of course be excluded from the security definition. For instance, in all models, the attacker is not allowed to simultaneously reveal the session key of an oracle π_i^s, and then ask a Test query to π_i^s, as this would trivially allow the adversary to correctly answer the Test query with probability 1. Moreover, the attacker must also not learn *both* the long-lived secret key (Corrupt) *and* the randomness (RevealRand) of an oracle involved in the Test-session, because then the attacker would learn the entire internal state of this oracle, and thus would be able to re-compute everything the oracle is able to compute, including the secret session key.

Research Challenges. The strongest form of security that is possible to achieve in such a model is to allow corruptions and randomness reveals even against oracles involved in the Test-session, provided that the attacker does not reveal *both* the randomness and the long-term secret of one oracle. (Corruptions of parties are of course only allowed *after* the key has been established, as

otherwise trivial man-in-the-middle attacks are possible.) *Is it possible to construct an ORKE protocol that achieves security in such a strong model?*

If a party is corrupted, the adversary can impersonate this party *in the future*. In some cases, the adversary can also break the security of session keys that have been generated *in the past* (e.g. if RSA key transport is used). The property that session keys computed before the corruption remain secure is known as *perfect forward secrecy* (PFS) [14,16]. In reaction to a conjecture of Krawczyk that ORKE protocols could only achieve a weaker form of PFS [24], Cremers showed that full PFS is generally achievable for ORKE protocols [11]. However until now, none of the proposed ORKE protocols has this property. *Can we construct an ORKE protocol that achieves perfect forward secrecy in such a strong model as eCK?*

Contributions. In this paper, we make the following contributions:

- *Novel generic construction.* We give an intuitive, relatively simple and efficient construction of an ORKE protocol with provable security in a model that allows *all non-trivial combinations* of corrupt- and reveal-queries, even against the Test-session.
- *Non-DH ORKE instantiation.* Instantiating our protocol with the factoring based NIKE protocol by Freire *et al.* [15], this yields an ORKE protocol based on the hardness of factoring large integers. This provides an alternative to known constructions based on (decisional) Diffie-Hellman.
- *First ORKE with perfect forward security under standard assumptions.* Our protocol is the first one-round AKE protocol which provides perfect forward security without random oracles.
- *Well-established, general assumptions.* The construction is based on general assumptions, namely the existence of secure non-interactive key exchange (NIKE) protocols [9,15], (unique) digital signatures, and a pseudorandom function. For all building blocks we require standard security properties.
- *Security in the Standard Model.* The security analysis is completely in the standard model, that is, without resorting to the Random Oracle heuristic [2] and without relying on non-standard complexity assumptions.

The Advantages of Generic Constructions. From a theoretical point of view, generic constructions show relations and implications between different types of cryptographic primitives. From a practical point of view, a generic protocol construction based on abstract building blocks allows to instantiate the protocol with *arbitrary* concrete instantiations of these building blocks — provided that they meet the required security properties. For instance, in order to obtain a "*post-quantum*"-instantiation of our protocol, it suffices to construct a NIKE scheme, digital signatures, and a PRF with *post-quantum* security and plug these primitives into the generic construction.

A common disadvantage of generic constructions is that they tend to be significantly less efficient than direct constructions. However, when instantiated with the NIKE schemes from [15], our protocol is already efficient enough to be

deployed in practice. See Section 5 for an efficiency comparison to other ORKE protocols.

Practical Motivation of the Model. Most cryptographic protocols inherently require "good" (i.e., independent and uniform) randomness to achieve their security goals. The availability of "good" random coins is simply assumed in the theoretical security analysis. However in practice, there are many famous examples where a flawed (i.e., low-entropy) generation of random numbers has led to serious security flaws. These include, for instance, the Debian OpenSSL bug,[1] the results of Lenstra *et al.* [28] and Heninger *et al.* [17] on the distribution of public keys on the Internet, or the case of certified smart cards considered by Bernstein *et al.* [3].

In our security model we allow the attacker to learn the *full* randomness of each party. Thus, even if this randomness is completely predictable, the protocol still provides security — as long as the long-lived secret keys of all parties are generated with good, "secret" randomness.

2 Related Work

Authenticated Key Exchange. An important line of research on the field of authenticated key exchange protocols started with Bellare and Rogaway [1] (the *BR* model) and Canetti and Krawczyk [8] (the *CK* model). The *CK* model is usually used to analyze one-round protocols, where authentication and key negotiation is performed very efficiently by two parties, only sending one message per party. Examples of such one-round protocols are MQV [27], KEA [26,30], or NAXOS [25]. HMQV [23], SMQV [32] were proposed to meet stronger security definitions. A comparison of different variants of the *CK* model can be found in [10,35]. Most constructions are proven secure in the Random Oracle Model (ROM) [2], with only a few exceptions [5,31,33].

PFS and KCI attacks. Perfect forward secrecy (PFS) is an important security goal for key-exchange protocols. Loosely speaking, PFS guarantees the secrecy of older session keys, even when the parties long-term key is compromised. Krawczyk [24] conjectured that no one-round protocol with implicit authentication can achieve full PFS in a *CK*-type model and introduced the notion of weak PFS (wPFS); this conjecture was refuted by Cremers *et al.* [11]. A protocol is wPFS secure, if the session key is indistinguishable from a random key and the parties long-term key is compromised if the adversary was *passive* during the session key negotiation [24, Section 3.2]. Similar to [11], we define rules for the security game to model and prove (full) PFS. In our security definition, the party corresponding to the tested oracle is allowed to be corrupted before the session completes. The only restriction to the corruption of parties in the test session is that the *intended partner* of the tested oracle is uncorrupted until the tested oracle accepts.

[1] https://www.debian.org/security/2008/dsa-1571

Another security goal of AKE protocols is security against *key-compromise impersonation* (KCI) attacks [24]. In a KCI attack, an adversary corrupts a party A and is able to authenticate herself to A as some uncorrupted party B. Since in the *eCK* model the adversary is always allowed to corrupt some party and learn the session randomness of the matching session, security in the *eCK* model naturally brings security against KCI attacks.

eCK Models. The term "extended Canetti-Krawczyk model" (eCK) was first introduced in [25]. The main difference to the CK model is that the RevealState-query (which has to be specified for each protocol) is replaced with a different query, namely RevealEphemeralExponent (which is a meaningful definition only for DH-based protocols, or other protocols where ephemeral exponents appear). In subsequent publications, the eCK model was often slightly modified, such that it is difficult to speak of "the" eCK model.

The *eCK-PFS* Security model. In 2012 Cremers and Feltz introduced a variant of the extended Canetti-Krawczyk model to capture perfect forward security [11]. The major difference between the *eCK* and *eCK-PFS* security models is the definition of session identifiers. Cremers et al. introduced the notion of *origin sessions*, which solves technical problems with the session identifier definition from the original *eCK*-model [12].

We slightly enhanced the *eCK-PFS* model in order to better model PFS, by introducing an explicit counter of adversarial interactions as done by Jager et al. [20] for the *BR* security model. Thus, we have a clear order of events and we can formally validate if a party was corrupted *before* or after a session *accepted* another party as a communication partner.

3 Preliminaries

In this paragraph we will define non-interactive key exchange (NIKE) and one-round key exchange (ORKE) protocols and their security.

3.1 Secure Non-Interactive Key Exchange

Definition 1. *A non-interactive key exchange (NIKE) scheme consists of two deterministic algorithms* (NIKEgen, NIKEkey).

NIKEgen$(1^\lambda, r)$ *takes a security parameter* λ *and randomness* $r \in \{0,1\}^\lambda$. *It outputs a key pair* (pk, sk). *We write* $(pk, sk) \xleftarrow{\$}$ NIKEgen(1^λ) *to denote that* NIKEgen$(1^\lambda, r)$ *is executed with uniformly random* $r \xleftarrow{\$} \{0,1\}^\lambda$.

NIKEkey(sk_i, pk_j) *is a* deterministic *algorithm which takes as input a secret key* sk_i *and a public key* pk_j, *and outputs a key* $k_{i,j}$.

We say that a NIKE scheme is correct, *if for all* $(pk_i, sk_i) \xleftarrow{\$}$ NIKEgen(1^λ) *and* $(pk_j, sk_j) \xleftarrow{\$}$ NIKEgen(1^λ) *holds that* NIKEkey$(sk_i, pk_j) =$ NIKEkey(sk_j, pk_i).

A NIKE scheme is used by d parties $P_1, \ldots, P_d$ as follows. Each party P_i generates a key pair $(pk_i, sk_i) \leftarrow \mathsf{NIKEgen}(1^\lambda)$ and publishes pk_i. In order to compute the key shared by P_i and P_j, party P_i computes $k_{i,j} = \mathsf{NIKEkey}(sk_i, pk_j)$. Similarly, party P_j computes $k_{j,i} = \mathsf{NIKEkey}(sk_j, pk_i)$. Correctness of the NIKE scheme guarantees that $k_{i,j} = k_{j,i}$.

CKS-light security. The *CKS-light* security model for NIKE protocols is relatively simplistic and compact. We choose this model because other (more complex) NIKE security models like *CKS*, *CKS-heavy* and *m-CKS-heavy* are polynomial-time equivalent to *CKS-light*. See [15] for more details.

Security of a NIKE protocol NIKE is defined by a game **NIKE** played between an adversary $\mathcal{A}$ and a challenger. The challenger takes a security parameter λ and a random bit b as input and answers all queries of $\mathcal{A}$ until she outputs a bit b'. The challenger answers the following queries for $\mathcal{A}$:

- $\mathsf{RegisterHonest}(i)$. $\mathcal{A}$ supplies an index i. The challenger runs $\mathsf{NIKEgen}(1^\lambda)$ to generate a key pair (pk_i, sk_i) and records the tuple $(\texttt{honest}, pk_i, sk_i)$ for later and returns pk_i to $\mathcal{A}$. This query may be asked *at most twice* by $\mathcal{A}$.
- $\mathsf{RegisterCorrupt}(pk_i)$. With this query $\mathcal{A}$ supplies a public key pk_i. The challenger records the tuple $(\texttt{corrupt}, pk_i)$ for later.
- $\mathsf{GetCorruptKey}(i, j)$. $\mathcal{A}$ supplies two indexes i and j where pk_i was registered as corrupt and pk_j as honest. The challenger runs $k \leftarrow \mathsf{NIKEkey}(sk_j, pk_i)$ and returns k to $\mathcal{A}$.
- $\mathsf{Test}(i, j)$. The adversary supplies two indexes i and j that were registered honestly. Now the challenger uses bit b: if $b = 0$, then the challenger runs $k_{i,j} \leftarrow \mathsf{NIKEkey}(pk_i, sk_j)$ and returns the key $k_{i,j}$. If $b = 1$, then the challenger samples a random element from the key space, records it for later, and returns the key to $\mathcal{A}$.

The game **NIKE** outputs 1, denoted by $\mathbf{NIKE}^{\mathcal{A}}_{\mathsf{NIKE}}(\lambda) = 1$ if $b = b'$ and 0 otherwise. We say $\mathcal{A}$ wins the game if $\mathbf{NIKE}^{\mathcal{A}}_{\mathsf{NIKE}}(\lambda) = 1$.

Definition 2. *For any adversary $\mathcal{A}$ playing the above* **NIKE** *game against a NIKE scheme* NIKE*, we define the advantage of winning the game* **NIKE** *as*

$$\boldsymbol{Adv}^{\mathit{CKS\text{-}light}}_{\mathsf{NIKE}}(\mathcal{A}) = \mathsf{Pr}\left[\mathbf{NIKE}^{\mathcal{A}}_{\mathsf{NIKE}}(\lambda) = 1\right] - \frac{1}{2}$$

Let λ be a security parameter, NIKE *be a NIKE protocol and $\mathcal{A}$ an adversary. We say* NIKE *is a CKS-light-*secure *NIKE protocol, if for all probabilistic polynomial-time adversaries $\mathcal{A}$, the function $\boldsymbol{Adv}^{\mathit{CKS\text{-}light}}_{\mathsf{NIKE}}(\mathcal{A})$ is a negligible function in λ.*

3.2 One-Round Key Exchange Protocols

Definition 3. *A one-round key exchange (ORKE) scheme consists of three deterministic algorithms* $(\mathsf{ORKEgen}, \mathsf{ORKEmsg}, \mathsf{ORKEkey})$.

- $\mathsf{ORKEgen}(1^\lambda, r)$ *takes a security parameter* λ *and randomness* $r \in \{0,1\}^\lambda$. *It outputs a key pair* (pk, sk). *We write* $(pk, sk) \xleftarrow{\$} \mathsf{ORKEgen}(1^\lambda)$ *to denote that* $\mathsf{ORKEgen}$ *is executed with uniformly random* $r \xleftarrow{\$} \{0,1\}^\lambda$.
- $\mathsf{ORKEmsg}(r_i, sk_i, pk_j)$ *takes as input randomness* $r_i \in \{0,1\}^\lambda$, *secret key* sk_i *and a public key* pk_j, *and outputs a message* m_i.
- $\mathsf{ORKEkey}(sk_i, pk_j, r_i, m_j)$ *takes as input a secret key* sk_i, *a public key* pk_j, *randomness* r_i, *and message* m_j. *It outputs a key* k.

We say that a ORKE scheme is correct, *if for all* $(pk_i, sk_i) \xleftarrow{\$} \mathsf{ORKEgen}(1^\lambda)$ *and* $(pk_j, sk_j) \xleftarrow{\$} \mathsf{ORKEgen}(1^\lambda)$, *and for all* $r_i, r_j \xleftarrow{\$} \{0,1\}^\lambda$ *holds that*

$$\mathsf{ORKEkey}(sk_i, pk_j, r_i, m_j) = \mathsf{ORKEkey}(sk_j, pk_i, r_j, m_i),$$

where $m_i := \mathsf{ORKEmsg}(r_i, sk_i, pk_j)$ *and* $m_j := \mathsf{ORKEmsg}(r_j, sk_j, pk_i)$.

A ORKE scheme is used by d parties $P_1, \ldots, P_d$ as follows. Each party P_i generates a key pair $(pk_i, sk_i) \xleftarrow{\$} \mathsf{ORKEgen}(1^\lambda)$ and publishes pk_i. Then, two parties P_i, P_j can establish a shared key as follows (see Figure 1 for an illustration).

1. P_i chooses $r_i \xleftarrow{\$} \{0,1\}^\lambda$, computes $m_i := \mathsf{ORKEmsg}(r_i, sk_i, pk_j)$, and sends m_i to P_j.
2. P_j chooses $r_j \xleftarrow{\$} \{0,1\}^\lambda$, computes $m_j := \mathsf{ORKEmsg}(r_j, sk_j, pk_i)$, and sends m_j to P_i.
   (Both messages m_i and m_j may be sent simultaneously, as this is a *one-round* protocol).
3. The shared key is computed by party P_i as $k_{i,j} := \mathsf{ORKEkey}(sk_i, pk_j, r_i, m_j)$. Similarly, party P_j computes $k_{j,i} = \mathsf{ORKEkey}(sk_j, pk_i, r_j, m_i)$. Correctness of the ORKE scheme guarantees that $k_{i,j} = k_{j,i}$.

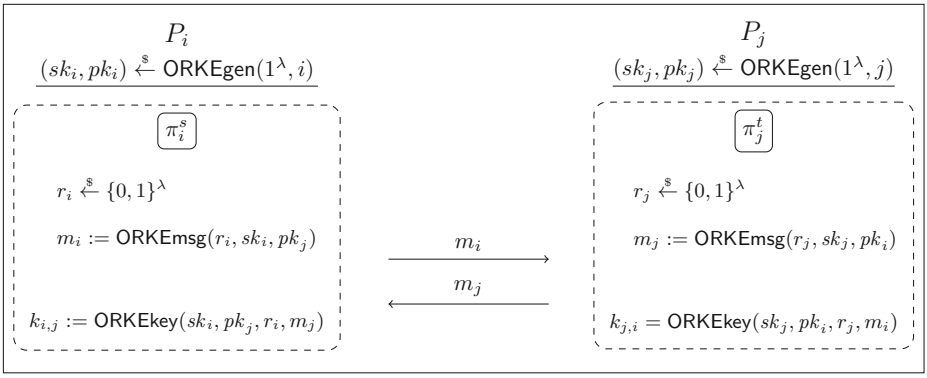

Fig. 1. Execution of an ORKE protocol

3.3 Secure One-Round Key Exchange

Security models for one-round key exchange have two major building blocks. The first defines the *execution environment* provided to an attacker on the AKE protocol. The second defines the rules of the game and the winning condition for an attacker.

Execution Environment. Consider a set of parties $\{P_1, \ldots, P_d\}$, $d \in \mathbb{N}$, where each party $P_i \in \{P_1, \ldots, P_d\}$ is a (potential) protocol participant and has a long-term key pair (pk_i, sk_i). To formalize several sequential and parallel executions of the protocol, each party P_i is modeled by a collection of ℓ oracles. Each oracle represents a process that executes one single instance of the protocol. All oracles representing party P_i have access to the same long-term key pair (pk_i, sk_i) of P_i and to all public keys $pk_1, \ldots, pk_d$. Moreover, each oracle π_i^s maintains as internal state the following variables:

- $\mathsf{Accepted}_i^s \in \mathbb{N} \cup \{\texttt{reject}\}$. This variable indicates whether and when the oracle accepted. It is initialized to $\mathsf{Accepted}_i^s = \texttt{reject}$.
- $\mathsf{Key}_i^s \in \mathcal{K} \cup \{\emptyset\}$, where $\mathcal{K}$ is the keyspace of the protocol and $\emptyset$ is the empty string, initialized to $\mathsf{Key}_i^s = \emptyset$.
- $\mathsf{Partner}_i^s$ containing the intended communication partner. We assume that each party P_i is uniquely identified by its public key pk_i, and therefore use public keys as identities.[2] The variable is initialized to $\mathsf{Partner}_i^s = \emptyset$.
- A variable $\mathcal{M}_{\mathsf{out}}^{i,s}$ storing the message sent by an oracle and a variable $\mathcal{M}_{\mathsf{in}}^{i,s}$ storing the received protocol message. Both are initialized as $\mathcal{M}_{\mathsf{in}}^{i,s} = \mathcal{M}_{\mathsf{out}}^{i,s} = \emptyset$.
- A variable $\mathsf{Randomness}_i^s$, which contains a uniformly string from $\{0,1\}^\kappa$. This string corresponds to the local randomness of an oracle. It is never changed or modified by an oracle.
- Variables $\mathsf{RevealedKey}_i^s, \mathsf{Corrupted}_i \in \mathbb{N}$, which will be used to determine if and when a $\mathsf{RevealKey}$ or $\mathsf{Corrupt}$ query was asked to this oracle or the corresponding party was corrupted (see below for details). These variables are initialized as $\mathsf{RevealedKey}_i^s = \mathsf{Corrupted}_i = \infty$.

We will assume (for simplicity) that

$$\mathsf{Key}_i^s \neq \emptyset \iff \mathsf{Accepted}_i^s \in \mathbb{N}.$$

We assume the adversary controls the network. Thus she is able to generate, manipulate or delay messages. Furthermore, the adversary can learn session keys, parties' secret long term keys and even the session randomness in our model. Formally the adversary may interact with the execution environment by issuing the following queries.

[2] In practice, several keys may be assigned to one identity. There are other ways to determine identities, for instance by using certificates. However, this is out of scope of this paper.

- $\mathsf{Send}(i, s, m) \rightarrow m'$: The adversary sends message m to oracle π_i^s. Party P_i processes message m according to the protocol specification and its internal oracle state π_i^s, updates its state[3], and optionally outputs an outgoing message m'.

  There is a distinguished initialization message `ini` which allows the adversary to activate the oracle with certain information. In particular, the initialization message contains the identity P_j of the intended partner of this oracle.
- $\mathsf{RevealKey}(i, s)$: if this is the τ-th query issued by $\mathcal{A}$, then the challenger sets $\mathsf{RevealedKey}_i^s := \tau$ and responds with the contents of variable Key_i^s. Recall that $\mathsf{Key}_i^s \neq \emptyset$ iff $\mathsf{Accepted}_i^s \in \mathbb{N}$.
- $\mathsf{RevealRand}(i, s)$: the challenger responds with the contents of $\mathsf{Randomness}_i^s$.
- $\mathsf{Corrupt}(i, pk^*)$: if this is the τ-th query issued by $\mathcal{A}$ (in total), then the challenger sets the oracle state $\mathsf{Corrupted}_i := \tau$ and responds with sk_i. Moreover, the public key pk_i is replaced (globally) with the adversarially-chosen key pk^*.[4]
- $\mathsf{Test}(i, s)$: This query may be asked only once throughout the game, it is answered as follows. Let $k_1 := \mathsf{Key}_i^s$ and $k_0 \xleftarrow{\$} \mathcal{K}$. If $\mathsf{Accepted}_i^s \in \mathbb{N}$, the oracle flips a fair coin $b \xleftarrow{\$} \{0, 1\}$ and returns k_b. If $\mathsf{Accepted}_i^s = \texttt{reject}$ or if $\mathsf{Partner}_i^s = j$ and P_j is corrupted when Test is issued, terminate the game and output a random bit.

***eCK-PFS* Security Definition.** In the following we give the security definition for one-round key-exchange protocols in the extended Canetti-Krawczyk model with perfect forward security. Firstly we introduce the partnering definitions from Cremers and Feltz. Secondly we define the rules by which an adversary has to play the AKE game in the *eCK-PFS*-model. Finally we define the security for one-round key-exchange protocols in the model.

Definition 4 (Origin session). *Consider two parties P_i and P_j with oracles π_i^s and π_j^t. We say π_i^s has* origin session *π_j^t, if $\mathcal{M}_{\mathsf{in}}^{i,s} = \mathcal{M}_{\mathsf{out}}^{j,t}$, and denote this by $\pi_i^s \xleftarrow{\mathsf{os}} \pi_j^t$.*

Alternatively we could say π_j^t is an origin session *of π_i^s, if $\mathcal{M}_{\mathsf{in}}^{i,s} = \mathcal{M}_{\mathsf{out}}^{j,t}$.*

Using the concept of origin sessions, we can define matching sessions as a symmetric relation of origin sessions: two sessions match, if they are origin sessions to each other. We capture this in the following definition.

Definition 5 (Matching session). *Consider two parties P_i and P_j with oracles π_i^s and π_j^t. We say π_i^s has a* matching session *to π_j^t (and vice versa), if π_i^s is an origin session of π_j^t and π_j^t is an origin session of π_i^s.*

[3] In particular, if π_i^s accepts after the τ-th query, set $\mathsf{Accepted}_i^s = \tau$.

[4] Note, that the adversary does not 'take control' of oracles corresponding to a corrupted party. But he learns the long-term secret key, and can henceforth simulate these oracles.

The notions of origin and matching sessions will be used in Definition 6 to exclude trivial attacks from the security model: If $\mathsf{Test}(i,s)$ is asked, restrictions are imposed on oracle π_i^s itself, and on oracles and parties *from which the test oracle has received a message*. On the other hand, sessions and parties to which a message was sent from the test session do not necessary play any role in Definition 6, for example if the test session has no matching session.

AKE Game. Consider the following security experiment $\mathbf{AKE}_{\Pi}^{\mathcal{A}}(\lambda)$ played between a challenger $\mathcal{C}$ and an adversary $\mathcal{A}$. The challenger receives the security parameter λ as an input and sets up all protocol parameters (like long term keys generation etc.). $\mathcal{C}$ simulates the protocol Π and keeps track of all variables of the execution environment. The adversary interacts by issuing any combination of the above mentioned queries. At some point of time during the game, she asks the Test_i^s query and gets a key k_b, which is either the exchanged key or a random key as described in the previous section. She may continue asking queries and finally outputs a bit b'. The game **AKE** outputs 1, denoted by $\mathbf{AKE}_{\Pi}^{\mathcal{A}}(\lambda) = 1$ if $b = b'$ and 0 otherwise.

Definition 6 (*eCK-PFS*-rules). *$\mathcal{A}$ plays the **AKE** game by eCK-PFS-*rules, *if the following conditions hold simultaneously when she issues* $\mathsf{Test}(i,s)$*:*

- $\mathsf{Accepted}_i^s = \tau$ *with* $\tau \in \mathbb{N}$.
- *$\mathcal{A}$ did not ask both* $\mathsf{Corrupt}(i, pk^*)$ *and* $\mathsf{RevealRand}(i,s)$.
- *If π_i^s has an origin session π_j^t, then it does not hold that both* $\mathsf{Corrupted}_j \leq \tau$ *and $\mathcal{A}$ asked* $\mathsf{RevealRand}(j,t)$.
- *If π_i^s has no origin session but intended partner* $\mathsf{Partner}_i^s = j$*, then it does not hold that* $\mathsf{Corrupted}_j \leq \tau$.

When $\mathcal{A}$ terminates and outputs a bit b', it also holds that $\mathcal{A}$ did not ask $\mathsf{RevealKey}(i,s)$ *and (if π_i^s has a matching session to π_j^t)* $\mathsf{RevealKey}(j,t)$.

We say $\mathcal{A}$ wins the AKE game, if $\mathbf{AKE}_{\Pi}^{\mathcal{A}}(\lambda) = 1$.

Definition 7 (*eCK-PFS*-security). *We define the advantage of $\mathcal{A}$ winning this game playing by eCK-PFS-rules as*

$$\boldsymbol{Adv}_{\Pi}^{eCK\text{-}PFS}(\mathcal{A}) = \mathsf{Pr}\left[\boldsymbol{AKE}_{\Pi}^{\mathcal{A}}(\lambda) = 1\right] - \frac{1}{2}.$$

*Let λ be a security parameter, Π be an AKE protocol and $\mathcal{A}$ an adversary. We say Π is an eCK-*secure *AKE protocol, if it is correct and for all probabilistic polynomial-time adversaries $\mathcal{A}$ playing by eCK-PFS-rules, the function $\boldsymbol{Adv}_{\Pi}^{eCK\text{-}PFS}(\mathcal{A})$ is a negligible function in λ.*

Remark 1. Note that this security definition includes perfect-forward secrecy and security against KCI attacks.

3.4 Further Building Blocks

Digital signatures. A digital signature scheme consists of three polynomial-time algorithms $\mathsf{SIG} = (\mathsf{SIGgen}, \mathsf{SIGsign}, \mathsf{SIGvfy})$. The key generation algorithm $(sk, pk) \xleftarrow{\$} \mathsf{SIGgen}(1^\lambda)$ generates a public verification key pk and a secret signing key sk on input of security parameter λ. Signing algorithm $\sigma \xleftarrow{\$} \mathsf{SIGsign}(sk, m)$ generates a signature for message m. Verification algorithm $\mathsf{SIGvfy}(pk, \sigma, m)$ returns 1 if σ is a valid signature for m under key pk, and 0 otherwise.

Definition 8. *We say that* SIG *is deterministic, if* SIGsign *is deterministic.*

Consider the following security experiment played between a challenger $\mathcal{C}$ and an adversary $\mathcal{A}$.

1. The challenger generates a public/secret key pair $(sk, pk) \xleftarrow{\$} \mathsf{SIGgen}(1^\lambda)$, the adversary receives pk as input.
2. The adversary may query arbitrary messages m_i to the challenger. The challenger replies to each query with a signature $\sigma_i = \mathsf{SIGsign}(sk, m_i)$. Here i is an index, ranging between $1 \leq i \leq q$ for some $q \in \mathbb{N}$. Queries can be made adaptively.
3. Eventually, the adversary outputs a message/signature pair (m, σ).

Definition 9. *We define the advantage on an adversary $\mathcal{A}$ in this game as*

$$\boldsymbol{Adv}^{sEUF\text{-}CMA}_{\mathsf{SIG}}(\mathcal{A}) := \mathsf{Pr}\left[(m,\sigma) \xleftarrow{\$} \mathcal{A}^{\mathcal{C}(\lambda)}(pk) : \begin{array}{l}\mathsf{SIGvfy}(pk, m, \sigma) = 1,\\ (m,\sigma) \neq (m_i, \sigma_i)\ \forall i\end{array}\right]$$

SIG *is* strongly secure *against* existential forgeries under adaptive chosen-message attacks *(sEUF-CMA), if* $\boldsymbol{Adv}^{sEUF\text{-}CMA}_{\mathsf{SIG}}(\mathcal{A})$ *is a negligible function in* λ *for all probabilistic polynomial-time adversaries* $\mathcal{A}$.

Remark 2. Deterministic signatures with sEUF-CMA security can be constructed, for instance, from verifiable unpredictable or verifiable random functions with large input spaces [4,18,19,29].

Pseudorandom functions. A *pseudo-random function* is an algorithm PRF. This algorithm implements a deterministic function $z = \mathsf{PRF}(k, x)$, taking as input a key $k \in \{0,1\}^\lambda$ and some bit string x, and returning a string $z \in \{0,1\}^\mu$.

Consider the following security experiment played between a challenger $\mathcal{C}$ and an adversary $\mathcal{A}$.

1. The challenger samples $k \xleftarrow{\$} \{0,1\}^\lambda$ uniformly random.
2. The adversary may query arbitrary values x_i to the challenger. The challenger replies to each query with $z_i = \mathsf{PRF}(k, x_i)$. Here i is an index, ranging between $1 \leq i \leq q$ for some $q \in \mathbb{N}$. Queries can be made adaptively.
3. Eventually, the adversary outputs value x and a special symbol $\top$. The challenger sets $z_0 = \mathsf{PRF}(k, x)$ and samples $z_1 \xleftarrow{\$} \{0,1\}^\mu$ uniformly random. Then it tosses a coin $b \xleftarrow{\$} \{0,1\}$, and returns z_b to the adversary.

4. Finally, the adversary outputs a guess $b' \in \{0,1\}$.

The Adversary wins the game, if she outputs b' such that $b = b'$.

Definition 10. *We denote the advantage of an adversary $\mathcal{A}$ in winning this game as*

$$\boldsymbol{Adv}^{prf}_{\mathsf{PRF}}(\mathcal{A}) = \mathsf{Pr}\left[b = b' \text{ for } b' \xleftarrow{\$} \mathcal{A}^{\mathcal{C}(\lambda)}(1^\lambda)\right] - \frac{1}{2}$$

We say that PRF *is a* secure *pseudo-random function, if for all probabilistic polynomial time adversaries $\mathcal{A}$ $\boldsymbol{Adv}^{prf}_{\mathsf{PRF}}(\mathcal{A})$ is a negligible function in λ.*

4 Generic Construction of eCK-Secure Key Exchange

Let $\mathsf{SIG} = (\mathsf{SIGgen}, \mathsf{SIGsign}, \mathsf{SIGvfy})$ be a *deterministic* signature scheme, $\mathsf{NIKE} = (\mathsf{NIKEgen}, \mathsf{NIKEkey})$ be a NIKE scheme, and let PRF be a pseudo-random function. Let sort be an arbitrary function which takes as input two strings (m_i, m_j), and outputs them according to some order (e.g. lexicographically). That is,

$$\mathsf{sort}(m_i, m_j) := \begin{cases} (m_i, m_j), & \text{if} \quad m_i \leq m_j, \\ (m_j, m_i), & \text{if} \quad m_i > m_j, \end{cases}$$

where $\leq$ and $>$ are defined with respect to some (arbitrary) ordering. We construct an ORKE protocol $\Pi = (\mathsf{ORKEgen}, \mathsf{ORKEmsg}, \mathsf{ORKEkey})$ as follows (see also Figure 2).

$\mathsf{ORKEgen}(1^\lambda)$ computes key pairs for the NIKE and digital signature scheme, respectively, as $(pk_i^{\mathsf{nike}}, sk_i^{\mathsf{nike}}) \xleftarrow{\$} \mathsf{NIKEgen}(1^\lambda)$ and $(pk_i^{\mathsf{sig}}, sk_i^{\mathsf{sig}}) \xleftarrow{\$} \mathsf{SIGgen}(1^\lambda)$, and outputs

$$(pk_i, sk_i) := ((pk_i^{\mathsf{nike}}, pk_i^{\mathsf{sig}}), (sk_i^{\mathsf{nike}}, sk_i^{\mathsf{sig}}))$$

$\mathsf{ORKEmsg}(r_i, sk_i, pk_j)$ parses $sk_i = (sk_i^{\mathsf{nike}}, sk_i^{\mathsf{sig}})$. Then it samples $r_i \xleftarrow{\$} \{0,1\}^\lambda$ and runs the key generation algorithm $(pk_i^{\mathsf{tmp}}, sk_i^{\mathsf{tmp}}) \xleftarrow{\$} \mathsf{NIKEgen}(1^\lambda, r_i)$ to generate a key pair of the NIKE scheme. Then it computes a signature over pk_i^{tmp} as $\sigma_i \xleftarrow{\$} \mathsf{SIGsign}(sk_i^{\mathsf{sig}}, pk_i^{\mathsf{tmp}})$ and outputs the message $m_i := (pk_i^{\mathsf{tmp}}, \sigma_i)$.

$\mathsf{ORKEkey}(sk_i, (pk_j^{\mathsf{nike}}, pk_j^{\mathsf{sig}}), r_i, m_j)$ first parses its input as $m_j = (pk_j^{\mathsf{tmp}}, \sigma_j)$ and $sk_i = (sk_i^{\mathsf{nike}}, sk_i^{\mathsf{sig}})$. If

$$\mathsf{SIGvfy}(pk_j^{\mathsf{sig}}, pk_j^{\mathsf{tmp}}, \sigma_j) \neq 1,$$

then it outputs $\perp$. Otherwise it runs $(pk_i^{\mathsf{tmp}}, sk_i^{\mathsf{tmp}}) \xleftarrow{\$} \mathsf{NIKEgen}(1^\lambda, r_i)$ to re-compute sk_i^{tmp} from r_i. Finally it derives the key k as follows.

1. Compute $T := \mathsf{sort}(pk_i^{\mathsf{tmp}}, pk_j^{\mathsf{tmp}})$.

2. Compute

$$k_{\mathsf{nike,nike}} := \mathsf{PRF}(\mathsf{NIKEkey}(sk_i^{\mathsf{nike}}, pk_j^{\mathsf{nike}}), T), \tag{1}$$

$$k_{\mathsf{nike,tmp}} := \mathsf{PRF}(\mathsf{NIKEkey}(sk_i^{\mathsf{nike}}, pk_j^{\mathsf{tmp}}), T), \tag{2}$$

$$k_{\mathsf{tmp,nike}} := \mathsf{PRF}(\mathsf{NIKEkey}(sk_i^{\mathsf{tmp}}, pk_j^{\mathsf{nike}}), T), \tag{3}$$

$$k_{\mathsf{tmp,tmp}} := \mathsf{NIKEkey}(sk_i^{\mathsf{tmp}}, pk_j^{\mathsf{tmp}}). \tag{4}$$

3. Compute k as

$$k := k_{\mathsf{nike,nike}} \oplus k_{\mathsf{nike,tmp}} \oplus k_{\mathsf{tmp,nike}} \oplus k_{\mathsf{tmp,tmp}}$$

and output k.

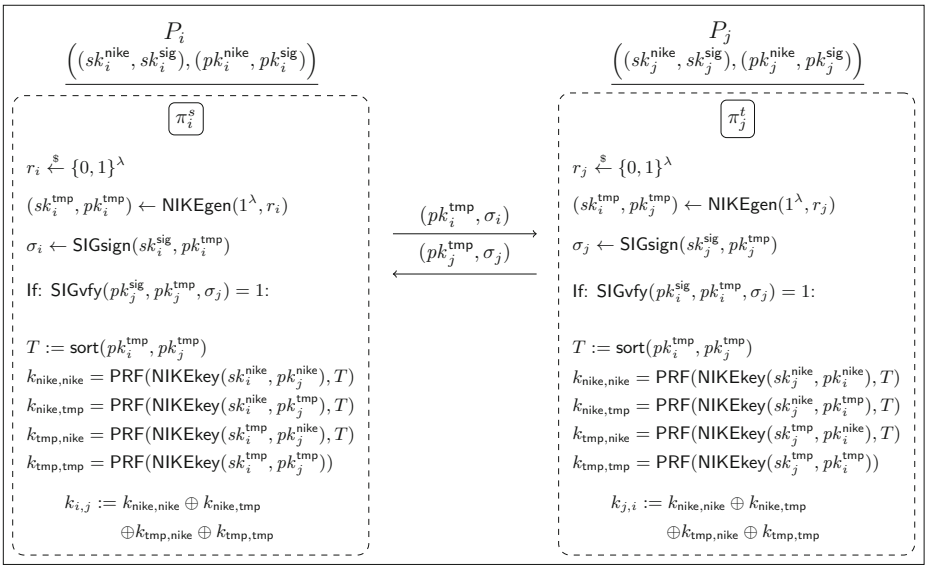

Fig. 2. Execution of protocol Π

Remark 3. In our generic construction Π we use a deterministic, strong existentially-unforgeable (*sEUF-CMA*) signature scheme. We could use a probabilistic signature scheme instead, but in this case we require a strong existentially-unforgeable *public coin* signature scheme.

The reason why we need *strong* existential unforgeability is the strictness of the matching conversation definition, which is also discussed in [7]. When using a probabilistic signature scheme, then we would need the public coin property to simulate RevealRand queries.

Even though such signatures may be easier or more efficiently to construct, we would not gain a better understanding of the reduction. Only the proofs would become harder to follow. For this reason we decided to use a deterministic scheme for simplicity.

Theorem 1. *From each attacker $\mathcal{A}$, we can construct attackers $\mathcal{B}_{\mathsf{sig}}$, $\mathcal{B}^{(1)}_{\mathsf{nike}}$, $\mathcal{B}^{(0)}_{\mathsf{nike}}$, and $\mathcal{B}_{\mathsf{prf}}$ such that*

$$\begin{aligned}\boldsymbol{Adv}^{eCK}_{\Pi}(\mathcal{A}) \leq\ & 4 \cdot d^2\ell^2 \cdot \left(\boldsymbol{Adv}^{CKS\text{-}light}_{\mathsf{NIKE}}(\mathcal{B}^{(1)}_{\mathsf{nike}}) + \boldsymbol{Adv}^{prf}_{\mathsf{PRF}}(\mathcal{B}_{\mathsf{prf}})\right) \\ & + 4 \cdot d \cdot \boldsymbol{Adv}^{sEUF\text{-}CMA}_{\mathsf{SIG}}(\mathcal{B}_{\mathsf{sig}}) + 4 \cdot \boldsymbol{Adv}^{CKS\text{-}light}_{\mathsf{NIKE}}(\mathcal{B}^{(0)}_{\mathsf{nike}})\end{aligned}$$

The running time of $\mathcal{B}_{\mathsf{sig}}$, $\mathcal{B}^{(1)}_{\mathsf{nike}}$, $\mathcal{B}^{(0)}_{\mathsf{nike}}$, and $\mathcal{B}_{\mathsf{prf}}$ is equal to the running time of $\mathcal{A}$ plus a minor overhead for the simulation of the security experiment for $\mathcal{A}$.

In order to prove Theorem 1, we will distinguish between four different types of attackers. Without loss of generality, we assume that an attacker always asks a $\mathsf{Test}(i,s)$-query for some (i,s). (Otherwise it is impossible to have a non-zero advantage, as then all computations are independent of the bit b sampled by the Test-query.) We distinguish between the following four types of attackers.

1. A Type-RR-attacker never asks $\mathsf{RevealRand}(i,s)$. If there exists an oracle π^t_j such that $\pi^s_i \xleftarrow{\text{os}} \pi^t_j$, then it also never asks $\mathsf{RevealRand}(j,t)$.
2. A Type-RC-attacker never asks $\mathsf{RevealRand}(i,s)$. If there exists an oracle π^t_j such that $\pi^s_i \xleftarrow{\text{os}} \pi^t_j$, then it also never asks $\mathsf{Corrupt}(j,\cdot)$.
3. A Type-CR-attacker never asks $\mathsf{Corrupt}(i,\cdot)$. If there exists an oracle π^t_j such that $\pi^s_i \xleftarrow{\text{os}} \pi^t_j$, then it also never asks $\mathsf{RevealRand}(j,t)$.
4. A Type-CC-attacker never asks $\mathsf{Corrupt}(i,\cdot)$. If there exists an oracle π^t_j such that $\pi^s_i \xleftarrow{\text{os}} \pi^t_j$, then it also never asks $\mathsf{Corrupt}(j,\cdot)$.

Note that each valid attacker in the sense of Definition 7 falls into (at least) one of these four categories. We will consider attackers of each type seperately in the sequel.

Intuition for the proof of Theorem 1. Let us now give some intuition why this classification of attackers will be useful for the security proof of Π. Recall that in protocol Π the key is computed as $k := k_{\mathsf{nike,nike}} \oplus k_{\mathsf{nike,tmp}} \oplus k_{\mathsf{tmp,nike}} \oplus k_{\mathsf{tmp,tmp}}$, where the keys $k_{\mathsf{nike,nike}}, k_{\mathsf{nike,tmp}}, k_{\mathsf{tmp,nike}}, k_{\mathsf{tmp,tmp}}$ are computed as described in Equations 1 to 4. The idea behind this construction is that in the proof we want to be able to reduce the indistinguishability of the ORKE-key k to the indistinguishability of a NIKE-key.

Recall that in the NIKE security experiment the attacker receives two challenge public-keys $pk^{\mathsf{nike}}, pk^{\mathsf{nike}\prime}$ from the challenger. In the reduction, we want to embed these keys into the view of the ORKE-attacker, such that we can embed the NIKE-challenge key into k while at the same time being able to answer all queries of the ORKE-attacker, in particular all $\mathsf{Corrupt}$ and $\mathsf{RevealRand}$ queries.

A Type-RR-attacker never asks $\mathsf{RevealRand}(i, s)$ and $\mathsf{RevealRand}(j, t)$ (if applicable). Thus, when considering Type-RR-attackers, then we can embed the NIKE-keys obtained from the NIKE-challenger as

$$pk_i^{\mathsf{tmp}} := pk^{\mathsf{nike}} \quad \text{and} \quad pk_j^{\mathsf{tmp}} := pk^{\mathsf{nike}'},$$

where pk_i^{tmp} and pk_j^{tmp} are the ephemeral keys generated by oracles π_i^s and π_j^t, respectively. Moreover, we embed the NIKE-challenge key k_{nike} as $k_{\mathsf{tmp,tmp}} := k_{\mathsf{nike}}$.

However, this embedding strategy does not work for Type-RC-attackers, because such an attacker might ask $\mathsf{RevealRand}(j, t)$. However, we know that a Type-RC attacker never asks a $\mathsf{Corrupt}(j, \cdot)$, therefore we are able to embed the NIKE challenge public keys as

$$pk_i^{\mathsf{tmp}} := pk^{\mathsf{nike}} \quad \text{and} \quad pk_j^{\mathsf{nike}} := pk^{\mathsf{nike}'},$$

where pk_i^{tmp} is the ephemeral keys generated by π_i^s, and pk_j^{nike} is the long-term secret of party P_j. The NIKE-challenge key k_{nike} is in this case embedded as $k_{\mathsf{tmp,nike}} := \mathsf{PRF}(k_{\mathsf{nike}}, T)$. The additional PRF is necessary in this case, because the embedding involves a long-term secret of one party of the test session. This long-term secret is used in (potentially) many protocol executions involving party P_j. Similarly, CR- and CC-type attackers can be handled by embedding the NIKE challenge public- and session as appropriate for each case.

Thus, the four different types of attackers correspond exactly to all four possible combinations of $\mathsf{Corrupt}$- and $\mathsf{RevealRand}$-queries against the Test-session that the attacker is allowed (resp. not allowed) to ask in our security model.

The full proof of Theorem 1 can be found in Appendix A.

5 Efficiency Comparison with Other ORKE Protocols

In Table 1 we compare the efficiency of instantiations of our construction to other one-round key-exchange protocols. We count the number of exponentiations and pairing evaluations. We do not distinguish an exponentiation in a DH group from an exponentiation in an RSA group.

We see that our generic construction ORKE, if instantiated with the most efficient NIKE primitive from [15], will be almost as efficient as the NAXOS protocol if the Cremers-Feltz compiler is applied [11]. The efficient NIKE primitive is secure in the random oracle model, but its security is based on the factoring problem.

The very high number of pairing evaluations within the standard model instantiation results from the fact, that the underlying NIKE scheme needs 3 pairing evaluations for key computation and we have to compute 4 NIKE keys per key-exchange at each party.

Table 1. Efficiency comparison of popular one-round key-exchange protocols to our generic construction.
[1] A variant of the Bellare-Rogaway model [1] with modified partnering definition. No ephemeral states can be revealed.
[2] The NAXOS protocol after application of the Cremers-Feltz compiler [11].
[3] Our construction instantiated with a secure NIKE scheme in the random-oracle model.
[4] Our construction instantiated with a standard-model NIKE scheme

	Standard Model	PFS	weak PFS	KCI	exp. per party	pairing evaluations	Security model
$\mathcal{TS}1$ [21]	✗	✗	✗	✗	1	-	BR^1
$\mathcal{TS}3$ [21]	✓	✓	✓	✗	3	-	BR^1
MQV	✗	✗	✓	✗	1	-	*CK*
HMQV	✗	✗	✓	✓	2	-	*CK*
KEA	✗	✗	✓	✓	2	-	*CK*
P1 [6]	✓	✗	✗	✓	8	2	*CK*
P2 [6]	✓	✗	✓	✓	10	2	*CK*
NAXOS	✗	✗	✓	✓	4	-	*eCK*
Okamoto	✓ $+\pi$PRF	✗	✓	✓	8	-	*eCK*
NAXOS^2_{pfs}	✗	✓	✓	✓	4	-	*eCK-PFS*
ORKE^3	✗ (NIKE)	✓	✓	✓	5	-	*eCK-PFS*
ORKE^4	✓	✓	✓	✓	16	12	*eCK-PFS*

References

1. Bellare, M., Rogaway, P.: Entity authentication and key distribution. In: Stinson, D.R. (ed.) CRYPTO 1993. LNCS, vol. 773, pp. 232–249. Springer, Heidelberg (1994)
2. Bellare, M., Rogaway, P.: Random oracles are practical: A paradigm for designing efficient protocols. In: Ashby, V. (ed.) ACM CCS 1993: 1st Conference on Computer and Communications Security, pp. 62–73, Fairfax, Virginia, USA, November 3–5. ACM Press (1993)
3. Bernstein, D.J., Chang, Y.-A., Cheng, C.-M., Chou, L.-P., Heninger, N., Lange, T., van Someren, N.: Factoring RSA keys from certified smart cards: Coppersmith in the wild. In: Sako, K., Sarkar, P. (eds.) ASIACRYPT 2013, Part II. LNCS, vol. 8270, pp. 341–360. Springer, Heidelberg (2013)
4. Boneh, D., Montgomery, H.W., Raghunathan, A.: Algebraic pseudorandom functions with improved efficiency from the augmented cascade. In: Al-Shaer, E., Keromytis, A.D., Shmatikov, V. (eds.) ACM CCS 2010: 17th Conference on Computer and Communications Security, pp. 131–140, Chicago, Illinois, USA, October 4–8. ACM Press (2010)
5. Boyd, C., Cliff, Y., Nieto, J.M.G., Paterson, K.G.: Efficient one-round key exchange in the standard model. In: Mu, Y., Susilo, W., Seberry, J. (eds.) ACISP 2008. LNCS, vol. 5107, pp. 69–83. Springer, Heidelberg (2008)
6. Boyd, C., Cliff, Y., Nieto, J.M.G., Paterson, K.G.: One-round key exchange in the standard model. IJACT **1**(3), 181–199 (2009)

7. Brzuska, C., Smart, N.P., Warinschi, B., Watson, G.J.: An analysis of the EMV channel establishment protocol. In: Sadeghi, A.-R., Gligor, V.D., Yung, M. (eds.) ACM CCS 2013: 20th Conference on Computer and Communications Security, pp. 373–386, Berlin, Germany, November 4–8. ACM Press (2013)
8. Canetti, R., Krawczyk, H.: Analysis of key-exchange protocols and their use for building secure channels. In: Pfitzmann, B. (ed.) EUROCRYPT 2001. LNCS, vol. 2045, pp. 453–474. Springer, Heidelberg (2001)
9. Cash, D., Kiltz, E., Shoup, V.: The twin Diffie-Hellman problem and applications. In: Smart, N.P. (ed.) EUROCRYPT 2008. LNCS, vol. 4965, pp. 127–145. Springer, Heidelberg (2008)
10. Cremers, C.: Examining indistinguishability-based security models for key exchange protocols: The case of CK, CK-HMQV, and eCK. In: Cheung, B.S.N., Hui, L.C.K., Sandhu, R.S., Wong, D.S. (eds.) ASIACCS 2011: 6th Conference on Computer and Communications Security, pp. 80–91, Hong Kong, China, March 22–24. ACM Press (2011)
11. Cremers, C., Feltz, M.: Beyond eCK: Perfect forward secrecy under actor compromise and ephemeral-key reveal. In: Foresti, S., Yung, M., Martinelli, F. (eds.) ESORICS 2012. LNCS, vol. 7459, pp. 734–751. Springer, Heidelberg (2012)
12. Cremers, C.J.F.: Formally and practically relating the CK, CK-HMQV, and eCK security models for authenticated key exchange. Cryptology ePrint Archive, Report 2009/253 (2009). http://eprint.iacr.org/2009/253
13. Dierks, T., Rescorla, E.: The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246 (Proposed Standard), Updated by RFCs 5746, 5878, 6176, August 2008
14. Diffie, W., van Oorschot, P.C., Wiener, M.J.: Authentication and authenticated key exchanges. Des. Codes Cryptography **2**(2), 107–125 (1992)
15. Freire, E.S.V., Hofheinz, D., Kiltz, E., Paterson, K.G.: Non-interactive key exchange. In: Kurosawa, K., Hanaoka, G. (eds.) PKC 2013. LNCS, vol. 7778, pp. 254–271. Springer, Heidelberg (2013)
16. Günther, C.G.: An identity-based key-exchange protocol. In: Quisquater, J.-J., Vandewalle, J. (eds.) EUROCRYPT 1989. LNCS, vol. 434, pp. 29–37. Springer, Heidelberg (1990)
17. Heninger, N., Durumeric, Z., Wustrow, E., Alex Halderman, J.: Mining your ps and qs: Detection of widespread weak keys in network devices. In: Kohno, T. (ed.) Proceedings of the 21th USENIX Security Symposium, Bellevue, WA, USA, August 8–10, pp. 205–220. USENIX Association (2012)
18. Hohenberger, S., Waters, B.: Constructing verifiable random functions with large input spaces. In: Gilbert, H. (ed.) EUROCRYPT 2010. LNCS, vol. 6110, pp. 656–672. Springer, Heidelberg (2010)
19. Jager, T.: Verifiable random functions from weaker assumptions. Cryptology ePrint Archive, Report 2014/799 (2014). http://eprint.iacr.org/
20. Jager, T., Kohlar, F., Schäge, S., Schwenk, J.: On the security of TLS-DHE in the standard model. In: Safavi-Naini, R., Canetti, R. (eds.) CRYPTO 2012. LNCS, vol. 7417, pp. 273–293. Springer, Heidelberg (2012)
21. Jeong, I.R., Katz, J., Lee, D.-H.: One-round protocols for two-party authenticated key exchange. In: Jakobsson, M., Yung, M., Zhou, J. (eds.) ACNS 2004. LNCS, vol. 3089, pp. 220–232. Springer, Heidelberg (2004)
22. Kaufman, C., Hoffman, P., Nir, Y., Eronen, P., Kivinen, T.: Internet Key Exchange Protocol Version 2 (IKEv2). RFC 7296 (INTERNET STANDARD). Updated by RFC 7427, October 2014

23. Krawczyk, H.: HMQV: A high-performance secure Diffie-Hellman protocol. In: Shoup, V. (ed.) CRYPTO 2005. LNCS, vol. 3621, pp. 546–566. Springer, Heidelberg (2005)
24. Krawczyk, H.: HMQV: A high-performance secure Diffie-Hellman protocol. Cryptology ePrint Archive, Report 2005/176 (2005). http://eprint.iacr.org/2005/176
25. LaMacchia, B.A., Lauter, K., Mityagin, A.: Stronger security of authenticated key exchange. In: Susilo, W., Liu, J.K., Mu, Y. (eds.) ProvSec 2007. LNCS, vol. 4784, pp. 1–16. Springer, Heidelberg (2007)
26. Lauter, K., Mityagin, A.: Security analysis of KEA authenticated key exchange protocol. In: Yung, M., Dodis, Y., Kiayias, A., Malkin, T. (eds.) PKC 2006. LNCS, vol. 3958, pp. 378–394. Springer, Heidelberg (2006)
27. Law, L., Menezes, A., Minghua, Q., Solinas, J., Vanstone, S.: An efficient protocol for authenticated key agreement. Designs, Codes and Cryptography **28**(2), 119–134 (2003)
28. Lenstra, A.K., Hughes, J.P., Augier, M., Bos, J.W., Kleinjung, T., Wachter, C.: Public keys. In: Safavi-Naini, R., Canetti, R. (eds.) CRYPTO 2012. LNCS, vol. 7417, pp. 626–642. Springer, Heidelberg (2012)
29. Lysyanskaya, A.: Unique signatures and verifiable random functions from the DH-DDH separation. In: Yung, M. (ed.) CRYPTO 2002. LNCS, vol. 2442, pp. 597–612. Springer, Heidelberg (2002)
30. NIST. Skipjack and kea algorithm specifications (1998). http://csrc.nist.gov/groups/STM/cavp/documents/skipjack/skipjack.pdf
31. Okamoto, T.: Authenticated key exchange and key encapsulation in the standard model. In: Kurosawa, K. (ed.) ASIACRYPT 2007. LNCS, vol. 4833, pp. 474–484. Springer, Heidelberg (2007)
32. Sarr, A.P., Elbaz-Vincent, P., Bajard, J.-C.: A new security model for authenticated key agreement. In: Garay, J.A., De Prisco, R. (eds.) SCN 2010. LNCS, vol. 6280, pp. 219–234. Springer, Heidelberg (2010)
33. Yang, Z.: Efficient eCK-secure authenticated key exchange protocols in the standard model. In: Qing, S., Zhou, J., Liu, D. (eds.) ICICS 2013. LNCS, vol. 8233, pp. 185–193. Springer, Heidelberg (2013)
34. Ylonen, T., Lonvick, C.: The Secure Shell (SSH) Transport Layer Protocol. RFC 4253 (Proposed Standard). Updated by RFC 6668, January 2006
35. Yoneyama, K., Zhao, Y.: Taxonomical security consideration of authenticated key exchange resilient to intermediate computation leakage. In: Boyen, X., Chen, X. (eds.) ProvSec 2011. LNCS, vol. 6980, pp. 348–365. Springer, Heidelberg (2011)