On the Limits of Authenticated Key Exchange Security with an Application to Bad Randomness

Michèle Feltz[1] and Cas Cremers[2]

[1]Institute of Information Security, ETH Zurich, Switzerland, feltzm@inf.ethz.ch
[2]University of Oxford, United Kingdom, cas.cremers@cs.ox.ac.uk

Abstract

State-of-the-art authenticated key exchange (AKE) protocols are proven secure in game-based security models. These models have considerably evolved in strength from the original Bellare-Rogaway model. However, so far only informal impossibility results, which suggest that no protocol can be secure against stronger adversaries, have been sketched. At the same time, there are many different security models being used, all of which aim to model the strongest possible adversary. In this paper we provide the first systematic analysis of the limits of game-based security models. Our analysis reveals that different security goals can be achieved in different relevant classes of AKE protocols. From our formal impossibility results, we derive strong security models for these protocol classes and give protocols that are secure in them. In particular, we analyse the security of AKE protocols in the presence of adversaries who can perform attacks based on chosen randomness, in which the adversary controls the randomness used in protocol sessions. Protocols that do not modify memory shared among sessions, which we call stateless protocols, are insecure against chosen-randomness attacks. We propose novel stateful protocols that provide resilience even against this worst case randomness failure, thereby weakening the security assumptions required on the random number generator.

Keywords. authenticated key exchange (AKE), security models, impossibility results, stateless protocols, stateful protocols, bad randomness, chosen-randomness.

1 Introduction

Authenticated Key Exchange (AKE) protocols have been a core building block of secure systems since the invention of public-key cryptography. The first formal security model for evaluating the security of AKE protocols was introduced in 1993 by Bellare and Rogaway [5]. Since then, there has been a steady stream of AKE protocols and associated security models. The most influential additions and modifications to the original Bellare-Rogaway model incorporate further security guarantees or increase the capabilities of the adversary [6, 12, 22, 24]. However, few results exist on the exact limits of AKE security and, consequently, on strongest security guarantees achievable by AKE protocols. Most impossibility results in the literature are implicitly formulated as restrictions on the adversary's behavior with regard to the target session, and suggest that it is impossible to construct a protocol secure with respect to a less restricted adversary. These observations on strongest possible adversaries are often incorrect or only hold under unstated protocol restrictions. For example, the claim that the eCK model is the strongest possible model for analyzing two-message AKE protocols [13, 24, 25] was refuted in [18], proving that stronger guarantees than eCK security can be achieved for two-message protocols. Further, most prominent AKE security models are formally incomparable, as shown in [14, 15]; technically, there are nearly as many security models as there are protocol proposals. The sometimes subtle differences between the models are in fact critical for security, because they determine whether the

security model covers practically relevant attack scenarios. We argue that rigorous impossibility results for key exchange protocol design and subsequent systematic security models are still missing in the context of ever stronger AKE security guarantees.

In this paper we show that many of the restrictions posed on the behavior of the adversary in the models stem from unstated assumptions on the considered protocol class. Specifically, our impossibility results reveal a previously unstated assumption on the protocols analyzed in AKE security models: the protocols do not modify memory shared among sessions, i.e., they only modify session-specific memory. It turns out that lifting this assumption enables the design of protocols that are secure against stronger adversaries that control the randomness used in protocol sessions. In particular, our results enable us to weaken the security assumptions on the random number generator (RNG) used to produce session-specific randomness. This is particularly timely given the recently discovered security vulnerabilities that involve either flawed [1, 26, 28, 35] or weakened [29, 31, 32, 36] RNGs. In 2008, Bello discovered a randomness vulnerability in Debian's OpenSSL package; keys generated by the RNG of this package were predictable [1]. As a consequence, protocol sessions of the DHE variant of SSL, e. g., might have been compromised as attackers were able to predict the ephemeral secret keys used to establish the shared session key and hence to decrypt further communication between client and server [1, 35]. In August 2013, attackers took control of Bitcoin transactions due to flaws in Android's Java and OpenSSL RNG [26]. As multiple transactions were signed using the same randomness in the ECDSA signature generation, an attacker was able to recover the long-term secret signing key of the user initiating the transactions and perform transactions on its behalf. Further, it seems that the security of certain cryptographic systems such as RNGs has been deliberately weakened to, e. g., eavesdrop on private communication [31]. We introduce new security models that incorporate the worst-case scenario of adversaries controlling the randomness involved in protocol sessions and propose countermeasures against such attacks. Security in our new models implies security under repeated or predictable randomness.

Contributions. First, we perform the first systematic analysis of the limits of game-based AKE security. We identify several relevant protocol classes for which we provide formal impossibility results. That is, given an arbitrary protocol from a certain class, we specify attacks that can be applied to this protocol. Our impossibility results (a) clarify which security guarantees cannot be achieved by any protocol in the respective class, and (b) allow us to systematically develop strong security models for each class. We give for each class a concrete protocol that is secure in the corresponding strong model.

Second, some of the models that we derive from our impossibility results allow the adversary to perform chosen-randomness attacks even against the target session: these attacks go far beyond attacks covered in earlier security models. While stateless protocols fail to achieve security against such adversaries, we present stateful protocols, which modify memory that is shared among sessions, and achieve security against attacks based on choosing session-specific randomness. We thereby significantly reduce the assumptions needed from the RNG.

Third, our exploration of the limits of game-based security leads to a protocol hierarchy based on the constructed protocols. Our hierarchy highlights the security guarantees that can be achieved by each class, and provides a novel way of selecting an optimal trade-off between the type of protocol and the security it offers.

Related work. *Impossibility results.* In general, proposals for AKE protocols are motivated by either claiming stronger security guarantees or improved efficiency. This has resulted in a large number of works that suggest to provide security against the strongest possible adversary [10, 13, 19, 20, 23–25]. Until now, such comments mostly relied on informal impossibility results. For example, Krawczyk [22] sketched a generic perfect forward secrecy attack, for which he claimed that it breaks the security of any "implicitly authenticated" two-message AKE protocol. This attack has led to the statements that (a) no two-message protocol can provide PFS [23, p. 56], and (b) the eCK model capturing only weak perfect forward secrecy is the strongest possible

model for analyzing two-message AKE protocols [24, 25]. However, in [18], it is shown that two-message protocols can achieve PFS even for eCK-like adversaries.

Yang et al. state that no protocol can be secure against *reset-and-replay attacks* on the target session [34, p. 120]. In a reset-and-replay attack the adversary first sets the randomness of a session to the same randomness as used in a previous session of the same user and then replays messages to the session so that both sessions compute the same session key [34]. Based on their impossibility result, they design a security model in which reset attacks are disallowed on the target session. As we see in Section 5.2, their result only holds for a particular class of protocols, namely the class of stateless protocols, which do not modify memory that is shared among sessions. Since we consider a wider range of protocols, we arrive at the conclusion that there are protocols that are secure against reset-and-replay attacks on the target session. In particular, we construct variants of the NAXOS protocol [24] that are secure against such attacks.

Similarly, Boyd and González Nieto [10] show that one-round AKE protocols that do not provide message replay detection cannot achieve PFS if the adversary can also reveal session-specific randomness of the target session's peer. Their argument is based on a variant of Krawczyk's PFS attack [22]. They note that protocols that provide replay detection via timestamps or counters are not vulnerable to this attack. We show in Section 7 that there are one-round protocols that do not require replay detection and are only vulnerable to Boyd and González Nieto's [10] attack if the target session is activated with a message replayed from the first session of its peer. We construct a protocol that achieves security even under compromise of the target session's randomness and the actor of that session's long-term secret key as long as the randomness of at least one of the previous sessions of the same user has not been compromised.

Randomness failures. The first models addressing the leakage of session-specific information include the eCK model [24] and the CK model [12]. The eCK model considers an information-leaking RNG that leaks values after they have been generated, which is modelled via the query ephemeral-key. Intermediate protocol computations are assumed to be outside of the adversary's control. In contrast, the CK model considers long-term keys stored in secure memory (e.g., an HSM), whereas protocol computations are (partly) done in less-protected memory. The adversary has read-only access to the less-protected memory through a query session-state. However, in the vast majority of proofs in the CK model, the less-protected memory has been defined to contain exactly the randomness, thereby effectively modeling an information-leaking RNG. Unlike our work, CK and eCK do not consider predictable, failing, or compromised RNGs.

Yang et al. [34] first analyzed AKE security w.r.t. adversaries who can manipulate random values. They define two security models: Reset-1 and Reset-2. In the Reset-1 model the adversary controls the randomness of each session, with the restrictions that the adversary (a) does not issue corrupt queries to the actor and peer of the test session, and (b) the randomness used in the test and partner session is not used in any other session. Thus, the Reset-1 model captures neither weak perfect forward secrecy nor reset-and-replay attacks on the target session or its partner session. In contrast, the models that we develop in Section 5 and Section 6 capture reset attacks on the target session, reset attacks on its partner session, and weak perfect forward secrecy. The Reset-2 model captures repeated secret randomness in multiple sessions due to reset attacks, but no chosen-randomness attacks. Whereas the Reset-2 model captures weak perfect forward secrecy, it does not allow the adversary to perform reset attacks against either the target session or its partner session. In Section 5.3 we design a similar model to the Reset-2 model, which we call X_{AKE}. In the X_{AKE} model, the adversary is allowed to perform reset attacks even against the target session and its partner session. We show that security in a model that captures chosen-randomness attacks implies security in the X_{AKE} model. Yang et al. [34] provide a transformation that turns a protocol secure in the Reset-2 model into a protocol that is secure in both models, by replacing the randomness x used in the original protocol by $F_K(x)$, where F is a pseudorandom function family, and K is an additional long-term secret key.

Ristenpart and Yilek [30] show that virtual machine (VM) snapshots can lead to VM reset

attacks. As a countermeasure, they propose a framework for *hedging* cryptographic operations based on preprocessing potentially bad RNG-supplied randomness together with additional inputs with HMAC to provide pseudorandomness for the cryptographic operation; their framework uses hedging techniques for public-key encryption of Bellare et al. [3]. Hedging a cryptographic operation means designing it in such a way that, given good randomness, the operation provably achieves strong security goals, and, given bad randomness, the operation achieves weaker, but still meaningful, security goals [3]. Our models cover VM reset attacks on stateless protocols, i.e., resettable randomness, and we thereby address some of their future work.

Stateless and stateful AKE protocols. Most AKE protocols (e. g., HMQV [22], NAXOS [24], CMQV [33]) are stateless, i.e., they only modify session-specific memory, whereas the memory that is shared among sessions is invariant under protocol execution. Furthermore, the security of stateful AKE protocols, which update the memory that is shared among sessions during execution of the protocol, has not been considered in the context of randomness failures.

A few stateful protocols have been suggested. For example, Blake-Wilson et al. [7] propose to modify their Protocol 2 by concatenating the secret value that is used as the secret material to derive the session key with the value of a counter. We denote this new protocol by Protocol 2C. Instead of running the protocol each time a session key is required, a new session key is obtained by simply incrementing the counter and computing a new hash value [7]. The idea of using a counter variable is presented in the context of special applications for which it might not be desirable to run the protocol whenever a new session key has to be established. However, no security proof of Protocol 2C has been given. In Section 7.1 we prove the security of the CNX protocol, a stateful variant of the NAXOS protocol. The CNX protocol includes a global counter value, which is shared across the sessions of a user, as input to the hash function H_1 used in the computation of the outgoing messages.

2 Authenticated key exchange framework

In this section we define a framework to reason about the security of AKE protocols belonging to different classes against adversaries with diverse capabilities. This framework allows to express existing AKE security models such as the eCK model [24], the eCK^w model [18] and the eCK-PFS model [18] as well as extensions of these models that permit the adversary to choose the randomness used in protocol sessions.

2.1 Security model

Sessions and session-specific memory. Let $\mathcal{P}$ be a finite set of N binary strings representing user identifiers. Each user can execute multiple instances of an AKE protocol, called sessions, concurrently. We can uniquely identify specific sessions of a user by referring to the order in which they are created. Thus, the i-th session of user $\hat{P}$ is denoted by the tuple $(\hat{P}, i) \in \mathcal{P} \times \mathbb{N}$. These tuples are not used by the protocol, but allow the adversary to identify the sessions he created. We model each user by a probabilistic Turing machine. For each user $\hat{P}$, the state of its Turing machine consists of the memory contents of the user, where we differentiate between session-specific memory and user memory, which is shared among different sessions. We take an abstract view on the session-specific memory and assume that it can be separated into distinct named fields, referred to as variables and listed in Table 1. Some of these variables are set upon session creation, whereas others are set or updated during execution of the protocol. The next step to be executed by the protocol is stored in the variable *step*. Alternatively, this value could be stored in the variable *data*. We choose to store it in a separate variable for clarity. We say that a session s has accepted (or is completed) if the value of its *status* variable taking values in the set $\{\texttt{active}, \texttt{accepted}, \texttt{rejected}\}$ is `accepted`. We denote by st_s the session-specific

actor	the session's actor (the user running the session)
peer	the session's peer (the intended communication partner)
role	taken role; either $\mathcal{I}$ (initiator) or $\mathcal{R}$ (responder)
sent, *recv*	concatenation of all messages sent, respectively received, in the session
status	session status; either `active`, `accepted`, or `rejected`
key	key established in the session
rand	randomness used in the session
data	any additional session-specific or protocol-specific data
step	protocol step to be executed (in the session)

Table 1: Elements of session state

memory related to session s. The session-specific memory contains the session-specific variables of Table 1. Initially we assume that each session-specific variable is undefined, denoted by $\perp$.

User memory. The user memory of some user stores the user's long-term public/secret key pair, the public key of all other users $\hat{Q} \in \mathcal{P}$ as well as additional variables that might be required by the protocol. The information stored in the user memory is accessed and possibly updated by sessions of the user according to the protocol specification. In contrast to session-specific information, data stored in the user memory of some user $\hat{P}$ is shared among different sessions of the user $\hat{P}$. We denote by $st_{\hat{P}}$ the user memory of user $\hat{P} \in \mathcal{P}$.

Game state and game behaviour (see also [11]). The adversary, modeled as a probabilistic polynomial-time algorithm, interacts with the users in the set $\mathcal{P}$ within a game through queries in a set Q. The state of the game (or game state) contains session-specific state information st_s for all sessions s, user-specific information $st_{\hat{P}}$ for each user $\hat{P} \in \mathcal{P}$ as well as other information related to the game such as some bit that the adversary attempts to guess. The game behaviour, which we denote by Φ, describes how the game processes the queries in Q. More precisely, the game behaviour Φ is an algorithm taking as input the current state of the game GST, a query $q \in Q$, a protocol π, and a security parameter 1^k, and returning a new state GST' as well as a response $\mathsf{response} \in \{0,1\}^* \cup \{\perp, \star\}$ to the adversary's query q.

Definition 1 (h-message protocol)**.** *Let 1^k be the security parameter. An* h-message protocol *π, where h is the sum of the number of messages sent and received during a protocol session, consists of*

- *a set of domain parameters,*
- *a probabilistic polynomial-time key generation algorithm* KeyGen*, which takes as input the security parameter and outputs a public/secret key pair, and*
- *a deterministic polynomial-time algorithm Ψ executed by a user in a session. This algorithm takes as input the security parameter 1^k, the session-specific memory st_s of a session s, the user memory $st_{\hat{P}}$ of the actor $\hat{P}$ of session s, and a message $m \in \{0,1\}^*$, and outputs a triple of elements $(m', st'_s, st'_{\hat{P}})$, where $m' \in \{0,1\}^* \cup \{\star\}$ is a message, st'_s is an updated internal session state, and $st'_{\hat{P}}$ is an updated state of the user memory of user $\hat{P}$.*

If h is even, then the number of messages $m' \neq \star$ output by Ψ during a protocol session is $\frac{h}{2}$ for both roles initiator and responder. If h is odd, then the number of messages $m' \neq \star$ output by Ψ during a protocol session is $\frac{h+1}{2}$ for the initiator role and $\frac{h-1}{2}$ for the responder role.

The output of the key exchange algorithm Ψ (see Definition 1) may include the value $\star$ to indicate that the session does not generate an outgoing message.

Setup of the game. A setup algorithm *SetupG* is used to generate a set of a fixed number N of user identifiers, to set all session-specific variables to $\perp$, and to initialize the user memory of each user. The algorithm *SetupG* takes as input the protocol π and the security parameter 1^k, and outputs an initial game state GST_{init}. More precisely, the setup algorithm proceeds as follows:

1. generate a set $\mathcal{P} = \{\hat{P}_1, ..., \hat{P}_N\}$ of N distinct binary strings (representing user identifiers),
2. for all users $\hat{P} \in \mathcal{P}$: generate a long-term public/secret key pair $(\mathsf{pk}_{\hat{P}}, \mathsf{sk}_{\hat{P}})$ using algorithm KeyGen,
3. for all users $\hat{P} \in \mathcal{P}$: store the key pair $(\mathsf{pk}_{\hat{P}}, \mathsf{sk}_{\hat{P}})$ together with the set $\{(\hat{P}, \mathsf{pk}_{\hat{P}}) \mid \hat{P} \in \mathcal{P} \backslash \{\hat{P}\}\}$ in the user memory $st_{\hat{P}}$, and
4. initialize all other user-specific variables if such variables are used by the protocol.

Queries. The specification of some of the queries that we define below is similar to queries defined in the framework of Boyd et al. [9]. The public-info query, which was informally introduced in [11, p. 4], allows the adversary to obtain information that was generated during the setup phase of the game such as the users' identifiers and their public keys.

- public-info(). The query returns a set $\mathcal{L}$ of information which contains the set $\{(\hat{P}, \mathsf{pk}_{\hat{P}}) \mid \hat{P} \in \mathcal{P}\}$ as well as the initial values of all other variables stored in the user memory of each user, except for the users' long-term secret key, if such variables are used by the protocol.

The queries in the set $Q_{\mathsf{R}} = \{\mathsf{create}, \mathsf{send}\}$ model regular execution of the protocol.

- create$(\hat{P}, r[, \hat{Q}])$. The query models the creation of a new session s for the user with identifier $\hat{P}$. It requires that $\hat{P} \in \mathcal{P}, \hat{Q} \in \mathcal{P}$, and that $r \in \{\mathcal{I}, \mathcal{R}\}$; otherwise, it returns $\perp$. Session variables are initialized as

$$(s_{actor}, s_{role}, s_{sent}, s_{recv}, s_{status}, s_{key}, s_{step}) \leftarrow (\hat{P}, r, \epsilon, \epsilon, \texttt{active}, \perp, 1) \ .$$

  A bit string in $\{0,1\}^k$ is sampled uniformly at random and assigned to s_{rand}; we assume that all randomness required during the execution of session s is deterministically derived from s_{rand}. If the optional peer identifier $\hat{Q}$ is provided, the variable s_{peer} is set to $\hat{Q}$.
  The key exchange algorithm Ψ is executed on input $(1^k, st_s, st_{\hat{P}}, \epsilon)$. The algorithm returns a triple of elements $(m', st'_s, st'_{\hat{P}})$. We set $st_s \leftarrow st'_s$ and $st_{\hat{P}} \leftarrow st'_{\hat{P}}$. The query returns m'.
- send$(\hat{P}, i, m)$. The query models sending message m to the i'th session of user $\hat{P}$, which we denote by s. It requires that $s_{status} = \texttt{active}$; otherwise it returns $\perp$. The algorithm Ψ is run on input $(1^k, st_s, st_{\hat{P}}, m)$, and outputs a triple of elements $(m', st'_s, st'_{\hat{P}})$. We set $st_s \leftarrow st'_s$ and $st_{\hat{P}} \leftarrow st'_{\hat{P}}$. The query returns m'.

The queries in the set $Q_{\mathsf{C}} = \{\mathsf{session\text{-}key}, \mathsf{corrupt}, \mathsf{randomness}, \mathsf{cr\text{-}create}\}$ that we define next model the corruption of a user's secrets. The randomness query models the adversary's capability of learning the randomness s_{rand} of a particular session s. In contrast, the cr-create query models the adversary's capability of choosing the randomness used within a session. Note that we do not explicitly model repeated randomness, i. e. secret uniform bits that have been used in previous key exchange sessions. However, we show in Section 5.3 that security in the model that we present in Section 5.1 implies security in a similar model capturing repeated randomness. The significance of session-key queries is threefold. First, key exchange protocols are required to provide security against known-key attacks [27], which can be ensured through key independence among different sessions. Known-key attacks are captured in security models via session-key queries on non-matching sessions. Second, in an unknown-key share (UKS) attack, the adversary establishes two sessions which compute the same session key even if both sessions have different intended communication partners. The relevance of UKS attacks is discussed, e. g., in [8]. A formal definition of a UKS attack and how it is reflected in a security model via session-key queries is given in [16]. Third, session-key queries capture replay attacks combined with chosen-randomness attacks. In these replay attacks, the adversary causes two non-matching sessions to compute the same session key by setting the randomness of the second session to the same randomness as used in the first session and replaying the messages received by the first session to the second session [34].
In the definition of the queries session-key and randomness we denote the i'th session of user $\hat{P}$ by s.

- session-key$(\hat{P}, i)$. The query requires that $s_{status} = \texttt{accepted}$; otherwise, it returns $\perp$. The query returns the session key s_{key} of session s.
- corrupt$(\hat{P})$. If $\hat{P} \notin \mathcal{P}$, then S returns $\perp$. Otherwise the query returns the long-term secret key $\mathsf{sk}_{\hat{P}}$ of user $\hat{P}$.
- randomness$(\hat{P}, i)$. If $s_{status} \neq \perp$, then the randomness s_{rand} used in session s is returned. Otherwise, the query returns $\perp$.
- cr-create$(\hat{P}, r, rnd[, \hat{Q}])$. The query models the creation of a new session s, using randomness rnd chosen by the adversary, for the user $\hat{P}$. The query requires that $\hat{P} \in \mathcal{P}, \hat{Q} \in \mathcal{P}, rnd \in \{0,1\}^k$, and that $r \in \{\mathcal{I}, \mathcal{R}\}$; otherwise, it returns $\perp$. Session variables are initialized as

$$(s_{actor}, s_{role}, s_{sent}, s_{recv}, s_{status}, s_{key}, s_{rand}, s_{step}) \leftarrow (\hat{P}, r, \epsilon, \epsilon, \texttt{active}, \perp, rnd, 1) \ .$$

  If the optional peer identifier $\hat{Q}$ is provided, the variable s_{peer} is set to $\hat{Q}$.
  The key exchange algorithm Ψ is executed on input $(1^k, st_s, st_{\hat{P}}, \epsilon)$. The algorithm returns a triple $(m', st'_s, st'_{\hat{P}})$. We set $st_s \leftarrow st'_s$ and $st_{\hat{P}} \leftarrow st'_{\hat{P}}$. The query returns m'.

The set $Q_{\mathsf{noCR}} = Q_{\mathsf{R}} \cup (Q_{\mathsf{C}} \setminus \{\text{cr-create}\})$ contains all execution and corruption queries, except the query cr-create.

The notion of matching sessions specifies when two sessions are supposed to be intended communication partners. It is formalized below via matching conversations as in [18, 24].

Definition 2 (Matching sessions)**.** *Let π be an h-message protocol. We say that two sessions s and s' of π are* matching *if $s_{status} = s'_{status} = \texttt{accepted}$ and $s_{actor} = s'_{peer} \wedge s_{peer} = s'_{actor} \wedge s_{sent} = s'_{recv} \wedge s_{recv} = s'_{sent} \wedge s_{role} \neq s'_{role}$.*

We next define a parameterized family of AKE security models. The parameters for each model consist of a subset Q of the above adversary queries and a freshness predicate F, which restricts the adversary from performing certain combinations of queries.

Definition 3 (AKE security model)**.** *Let π be an h-message protocol. Let Q be a set of adversary queries such that $Q_{\mathsf{R}} \subseteq Q \subseteq Q_{\mathsf{R}} \cup Q_{\mathsf{C}}$. Let F be a freshness predicate, that is, a predicate that takes a session of protocol π and a sequence of queries (including arguments and results) in Q. We call (Q, F) an AKE security model.*

Remark 1. *In this work we fix a particular definition for matching sessions (namely, Definition 2) and construct strong security models with respect to this definition. It is straightforward to adapt these models to other definitions of matching sessions that are suitable for analyzing protocols such as (H)MQV that allow two sessions performing the same role to compute the same session key.*

2.2 Security experiment

We associate to each AKE security model $X = (Q, F)$ a security experiment $W(X)$, defined below, played by an adversary E against a challenger. To win the experiment, the adversary aims to distinguish a real session key from a random key, modelled through the following query.

- test-session(s). This query requires that $s_{status} = \texttt{accepted}$; otherwise, it returns $\perp$. A bit b is chosen at random. If $b = 0$, then s_{key} is returned. If $b = 1$, then a random key is returned according to the probability distribution of keys generated by the protocol.

Definition 4 (Security experiment $W(X)$)**.** *Let π be an h-message protocol. Let $X = (Q, F)$ be an AKE security model. We define experiment $W(X)$, between an adversary E and a challenger who implements all the users, as follows:*

1. *The game is initialized with domain parameters for security parameter 1^k and the setup algorithm SetupG is executed.*

2. *The adversary E first issues the query* public-info*, and then performs any sequence of queries from the set Q.*
3. *At some point in the experiment, E issues a* test-session *query to a session s that has accepted and satisfies F at the time the query is issued.*
4. *The adversary may continue with queries from Q, under the condition that the test session must continue to satisfy F.*
5. *Finally, E outputs a bit b' as his guess for b.*

The adversary E wins the security experiment $W(X)$ if he correctly guesses the bit b chosen by the challenger during the test-session query (i. e., if $b = b'$, where b' is E's guess). Success of E in the experiment is expressed in terms of E's advantage in distinguishing whether he received the real or a random session key in response to the test-session query. The advantage of adversary E in the above security experiment against a key exchange protocol π for security parameter k is defined as $Adv^{\pi,E}_{W(X)}(k) = |2P(b = b') - 1|$.

Definition 5 (AKE security)**.** *A key exchange protocol π is said to be* secure in AKE security model (Q, F) *if, for all PPT adversaries E, it holds that*

- *if two users successfully complete matching sessions, then they compute the same session key,*
- *the probability of event* Multiple-Match$^{W(X)}_{\pi,E}(k)$ *is negligible, where* Multiple-Match$^{W(X)}_{\pi,E}(k)$ *denotes the event that there exists a session that has accepted with at least two matching sessions, and*
- *E has no more than a negligible advantage in winning the $W(X)$ security experiment, that is, there exists a negligible function negl in the security parameter k such that $Adv^{\pi,E}_{W(X)}(k) \leq negl(k)$.*

Informally, the second requirement in Definition 5 (see also [5]) states that, for a given session of protocol π that has accepted, it holds that its matching session, if it exists, is unique.

3 Protocol Classes

In this section we define a series of relevant protocol classes. The largest protocol class includes, e. g., one-round protocols and Diffie-Hellman type protocols. As we see in Section 4 and Section 6, the distinction between these classes allows the systematic development of strong security models for analyzing protocols belonging to the respective classes.

3.1 Classes AKE, INDP, and INDP-DH

We start by defining a global class of AKE protocols. Such protocols are required to be executable, i. e., if the messages of two users $\hat{A}$ and $\hat{B}$ are faithfully relayed to each other, then both users end up with a shared session key (see also [4–6]). A second requirement ensures that protocol messages depend on session-specific randomness. Both properties are used for proving impossibility results in Section 4.

Definition 6 (Protocol class AKE)**.** *We define* AKE *as the class of all h-message protocols that meet the following requirements: In the presence of an eavesdropping adversary,*

- *two users $\hat{A}$ and $\hat{B}$ can complete matching sessions, in which case they hold the same session key, and*
- *the probability that two sessions of the same user output in all protocol steps identical messages is negligible in the security parameter.*

We next consider a subclass of two-message AKE protocols, namely the class INDP of one-round AKE protocols, where the outgoing message can be computed before any (valid) message is received. Formally, we define the class of one-round protocols as follows.

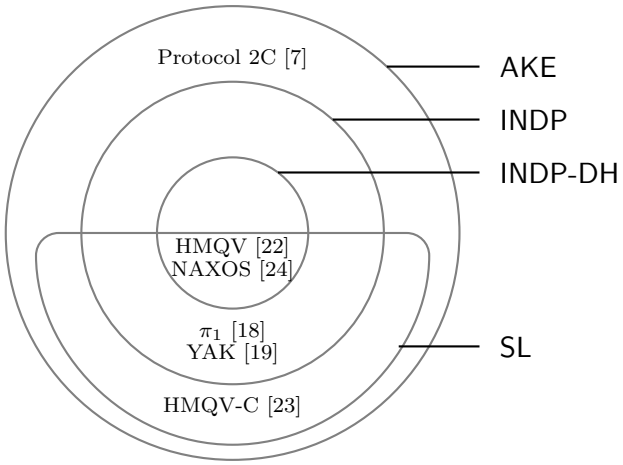

Figure 1: Venn Diagram of the protocol classes and example protocols

Definition 7 (Protocol class INDP)**.** *The protocol class* INDP *consists of all two-message protocols in* AKE *for which the outgoing message of any session s with status $s_{status} \neq$* `rejected` *does not depend on the incoming message.*

We define the class INDP-DH of one-round Diffie-Hellman type protocols as a subclass of INDP as follows.

Definition 8 (Protocol class INDP-DH)**.** *Let $G = \langle g \rangle$ be a cyclic group of prime order p generated by g. Let* KeyGen *be the key generation algorithm defined as* KeyGen()*: Choose $a \in_R [0, p-1]$; Set $A \leftarrow g^a$; Return secret key* $\mathsf{sk} = a$ *and public key* $\mathsf{pk} = A$*.*

The protocol class INDP-DH *consists of all protocols in the class* INDP *with domain parameters (G, g, p), key generation algorithm* KeyGen*, and where the outgoing message of any initiator session s in status $s_{status} \neq$* `rejected` *is of the form $(\delta, g^{f_{\mathcal{I}}(1^k, st_s, st_{\hat{A}})})$ and the outgoing message of any responder session s' in status $s'_{status} \neq$* `rejected` *is of the form $(\delta', g^{f_{\mathcal{R}}(1^k, st_{s'}, st_{\hat{B}})})$, where $\hat{A}$ is the actor of session s, $\hat{B}$ is the actor of session s', $f_{\mathcal{I}}, f_{\mathcal{R}} : \{0,1\}^* \to \mathbb{Z}_p$ are two polynomial-time computable functions, and δ, δ' is optional publicly available information such as the identifiers of actor and peer of the session.*

3.2 Stateless and stateful protocols

We distinguish between stateless and stateful protocols. Stateless protocols leave the state of a user's memory (i.e., the memory that is shared among sessions) invariant under execution of the protocol. In contrast, the state of a user's memory is modified when executing a protocol that is not stateless. Examples of stateless and stateful protocols are given in Figure 1 and in Section 7.

We define the class of stateless AKE protocols as a subclass of the class AKE as follows.

Definition 9 (Stateless protocol)**.** *Let $\mathcal{A}, \mathcal{B}$, and $\mathcal{C}$ be sets. Let* $\mathrm{proj}_3 : \mathcal{A} \times \mathcal{B} \times \mathcal{C} \to \mathcal{C}$ *be the map given by* $\mathrm{proj}_3(a, b, c) = c$ *for all $(a, b, c) \in \mathcal{A} \times \mathcal{B} \times \mathcal{C}$. Let π be a protocol in the class* AKE*. We say that π is a* stateless protocol *if*

$$\mathrm{proj}_3\big(\Psi(1^k, st_s, st_{\hat{P}}, m)\big) = st_{\hat{P}},$$

for all $(k, st_s, st_{\hat{P}}, m) \in \mathbb{N} \times \{st_s \mid s \in \mathcal{P} \times \mathbb{N}\} \times \{st_{\hat{P}} \mid \hat{P} \in \mathcal{P}\} \times \{0,1\}^*$. We denote by SL *the subclass of* AKE *containing all stateless protocols.*

Definition 10 (Stateful protocol)**.** *Let π be a protocol in the class* AKE*. We say that π is a* stateful protocol *if π is not stateless.*

Remark 2. *Stateless protocols cannot provide message replay detection to reject messages that have been received in earlier sessions of the same user as this would require storing all previously received messages in a table in the user memory and, upon receipt of a valid message in a session, accessing the table in the user memory and checking whether the message corresponds to a message in the table.*

Most recently proposed strong AKE protocols fall into the narrow class $\mathsf{INDP\text{-}DH} \cap \mathsf{SL}$. As we illustrate next, protocols from the wider classes can achieve stronger security.

4 Impossibility results and strong models for stateless protocols

In this section we provide impossibility results for protocols in the classes $\mathsf{AKE} \cap \mathsf{SL}$, $\mathsf{INDP} \cap \mathsf{SL}$ and $\mathsf{INDP\text{-}DH} \cap \mathsf{SL}$ with respect to an adversary who is given access to the queries in the set $Q_{\mathsf{noCR}} = Q_{\mathsf{R}} \cup (Q_{\mathsf{C}} \setminus \{\mathsf{cr\text{-}create}\})$. We then derive strong security models for reasoning about the security of protocols in the respective classes from these impossibility results. Each of our models can be satisfied by existing stateless protocols. In Section 5 we show that no stateless protocol can achieve security in stronger models, in which the adversary can additionally perform chosen-randomness attacks via the cr-create query.

We start by defining the notion of partially matching sessions in a similar way as matching conversations in [5, Definition 4.1].

Definition 11 (Partially matching sessions)**.** *Let π be an h-message protocol, where $h \geq 2$. Let s denote a session of π with $s_{status} = \texttt{accepted}$. We say that session s is* partially matching *session s' in status $s'_{status} \neq \perp$ if the following conditions hold:*

- $s_{role} \neq s'_{role} \wedge s_{actor} = s'_{peer} \wedge s_{peer} = s'_{actor}$ *and either*
- $s_{role} = \mathcal{I} \wedge s_{send}\,[1..m] = s'_{recv}\,[1..m] \wedge s_{recv}\,[1..m] = s'_{send}\,[1..m]$ *with $m = \frac{h}{2}$ if h is even and $m = \frac{h-1}{2}$ if h is odd, or*
- $s_{role} = \mathcal{R} \wedge s_{send}\,[1..(m-1)] = s'_{recv}\,[1..(m-1)] \wedge s_{recv}\,[1..m] = s'_{send}\,[1..m]$ *with $m = \frac{h}{2}$ if h is even and $m = \frac{h+1}{2}$ if h is odd,*

where $s_{send}\,[1..l]$ denotes the concatenation of the first l messages sent by session s and $s_{recv}\,[1..l]$ denotes the concatenation of the first l messages received by session s.

Remark 3. *In contrast to the notion of matching sessions (see Definition 2), which is only defined for completed sessions, the notion of partially matching sessions does not require the last message sent by one session to be received by the partner session. This allows us to capture combinations of* randomness *and* corrupt *attacks on such sessions as well.*

To relate a received message that was not constructed by the adversary to the session it originates from, we use the concept of origin-session, which was first introduced in [17]. The existence of an origin-session for a given session implies integrity of the received messages.

Definition 12 (origin-session)**.** *We say that a session s' with $s'_{status} \neq \perp$ is an origin-session for a session s with $s_{status} = \texttt{accepted}$ if $s'_{send} = s_{recv}$.*

Theorem 1 (Impossibility result for $\mathsf{AKE} \cap \mathsf{SL}$)**.** *Let π be an arbitrary protocol in the class $\mathsf{AKE} \cap \mathsf{SL}$. Let $X = (Q_{\mathsf{noCR}}, F)$ be the AKE security model with F being true for all sessions s and all sequences of queries. Let s^* denote the test session and let s' denote a session such that s^* is partially matching session s'. There exist adversaries who win the security experiment $W(X)$ against protocol π with non-negligible probability by issuing either*

1. *a query* session-key(s^*)*, or*
2. *a query* session-key$(\tilde{s})$*, where $\tilde{s}$ and s^* are matching sessions, or*
3. *a query* corrupt(s^*_{actor}) *and a query* randomness(s^*)*, or*

4. *a query* $\mathsf{corrupt}(s^*_{peer})$ *as well as a query* $\mathsf{randomness}(s')$*, or*
5. *a query* $\mathsf{corrupt}(s^*_{peer})$ *before creation of session* s^* *via a* create *query or as long as* $s^*_{status} =$ `active` *and* $s^*_{recv} = \epsilon$*, and impersonating the peer to session* s^**.*

Proof. Let $\pi \in \mathsf{AKE} \cap \mathsf{SL}$. There exist PPT adversaries who win the security game $W(X)$ against protocol π with non-negligible probability, as follows.

Scenario 1: Adversary E_1 establishes two sessions s and s' that are matching (via a sequence of create and send queries) and chooses session s as the test session. Issuing a $\mathsf{session\text{-}key}(s)$ query reveals the session key of session s.

Scenario 2: Since $\pi \in \mathsf{AKE}$, adversary E_2 can establish via a sequence of create and send queries two sessions s and s' that are matching according to Definition 2. He then issues the test-session query to one of the two sessions, say to session s, and issues a session-key query to session s'. E_2 thereby learns the session key of session s.

Scenario 3: Adversary E_3 establishes two sessions s and s' that are matching and chooses session s as the test session. Issuing the queries $\mathsf{corrupt}(s_{actor})$ and $\mathsf{randomness}(s)$ reveals the long-term secret key of $s_{actor} = \hat{P}$ and the randomness s_{rand} of session s, respectively. Together with the set $\mathcal{L}$ returned as response to the query public-info and the last message received during session s, which we denote by m', the adversary can compute the session key of session s by executing the algorithm Ψ on input $(1^k, st_s, st_{\hat{P}}, m')$. Note that the user state remains unchanged during protocol execution as $\pi \in \mathsf{SL}$.

Scenario 4: Assume that $\pi \in \mathsf{AKE} \cap \mathsf{SL}$ is an h-message protocol with h even. The proof works similarly for the case where h is odd.

Case 1: test session in initiator role. First E_4 establishes via a sequence of create and send queries an initiator session s and a responder session s' such that s and s' are matching sessions (this is possible since $\pi \in \mathsf{AKE}$). Clearly, session s is also partially matching session s'. He then chooses session s as the test session. Issuing the queries $\mathsf{corrupt}(s_{peer})$ and $\mathsf{randomness}(s')$ reveals the long-term secret key of $s_{peer} = \hat{P}$ and the randomness s'_{rand} of session s', respectively. Since $\pi \in \mathsf{SL}$ and the initial values of additional variables in the user memory are returned as response to the query public-info, the user state $st_{\hat{P}}$ is known to the adversary. Hence, the adversary can emulate the session key computation of a matching session and compute the session key of session s by executing Ψ on input $(1^k, st_{s'}, st_{\hat{P}}, m)$, where m denotes the last incoming message to session s'.

Case 2: test session in responder role. First E_4 establishes via a sequence of create and send queries an incomplete initiator session s and a responder session s' such that s' is partially matching session s (this is possible since $\pi \in \mathsf{AKE}$). He then chooses session s' as the test session. Issuing the queries $\mathsf{corrupt}(s'_{peer})$ and $\mathsf{randomness}(s)$ reveals the long-term secret key of s'_{peer} and the randomness of session s, respectively. Similar to the previous case, the adversary is able to compute the session key of session s'.

Scenario 5: Adversary E_5 issues a corrupt query to some user, say user $\hat{Q}$. He then creates a responder session s by issuing the query $\mathsf{create}(\hat{P}, \mathcal{R}, \hat{Q})$. The adversary now impersonates user $\hat{Q}$ to s_{actor} as follows. E_5 chooses randomness $r \in_R \{0,1\}^k$ and runs the protocol with $\hat{P}$ on behalf of $\hat{Q}$ by executing the algorithm Ψ. The algorithm Ψ executed by the adversary takes as input, among others, the user state of user $\hat{Q}$ containing its long-term secret key $\mathsf{sk}_{\hat{Q}}$ and the set $\mathcal{L}$ returned as response to the public-info query. Once session s has accepted, he chooses the latter as the test session. The adversary can compute the session key of session s, for which no origin-session exists, by emulating a matching session. Note that a similar attack also works against (a) an initiator session, (b) an initiator session s such that $s_{status} =$ `active` and $s_{recv} = \epsilon$, and (c) a responder session s such that $s_{status} =$ `active` and $s_{recv} = \epsilon$.

□

Theorem 1 gives rise to the security model $\Omega_{\mathsf{AKE}\cap\mathsf{SL}}$ defined as follows. The associated freshness notion restricts the adversary from performing the generic attacks specified in Theorem 1.

Definition 13 ($\Omega_{\mathsf{AKE}\cap\mathsf{SL}}$)**.** *The $\Omega_{\mathsf{AKE}\cap\mathsf{SL}}$ model is defined by (Q, F), where $Q = Q_{\mathsf{noCR}}$ and a session is said to satisfy F if all of the following conditions hold:*

1. *no* $\mathsf{session\text{-}key}(s)$ *query has been issued,*
2. *for all sessions s^* such that s^* matches s, no* $\mathsf{session\text{-}key}(s^*)$ *query has been issued,*
3. *not both queries* $\mathsf{corrupt}(s_{actor})$ *and* $\mathsf{randomness}(s)$ *have been issued,*
4. *for all sessions s' such that s is partially matching session s', not both queries* $\mathsf{corrupt}(s_{peer})$ *and* $\mathsf{randomness}(s')$ *have been issued, and*
5. *if there exists no origin-session for session s, then no* $\mathsf{corrupt}(s_{peer})$ *query has been issued before creation of session s via a* create *query or as long as $s_{status} = \mathtt{active}$ and $s_{recv} = \epsilon$.*

To see that there exist protocols in the class $\mathsf{AKE} \cap \mathsf{SL}$ that are secure in the model $\Omega_{\mathsf{AKE}\cap\mathsf{SL}}$, consider the protocol SIG*(NAXOS) obtained by applying the signature transformation SIG with optional fields from [18] to the NAXOS protocol. Clearly, protocol SIG*(NAXOS) does not belong to the class $\mathsf{INDP} \cap \mathsf{SL}$ since the outgoing message of a responder session depends on the Diffie-Hellman exponential contained in the incoming message. Adapting the proof of [18, Theorem 1], it can be shown that protocol SIG*(NAXOS) is secure in model $\Omega_{\mathsf{AKE}\cap\mathsf{SL}}$.

Even though security in model $\Omega_{\mathsf{AKE}\cap\mathsf{SL}}$ can be achieved by protocols in the class $\mathsf{AKE} \cap \mathsf{SL}$, we next show that no protocol in the class INDP can provide these strong security guarantees.

Proposition 1 (Impossibility result for INDP)**.** *No protocol in the class* INDP *can satisfy security in the model $\Omega_{\mathsf{AKE}\cap\mathsf{SL}}$.*

Proof. Let π be an arbitrary protocol in INDP. There exists an adversary E who wins the $W(\Omega_{\mathsf{AKE}\cap\mathsf{SL}})$ experiment against the challenger with non-negligible probability as follows. The adversary E first reveals the long-term secret key of some user $\hat{A}$ by issuing the query $\mathsf{corrupt}(\hat{A})$. Since E knows the values of all the variables stored in the user memory of user $\hat{A}$ (through the queries public-info and $\mathsf{corrupt}(\hat{A})$), E can run the first protocol execution step on behalf of $\hat{A}$ by executing the deterministic algorithm Ψ on input $(1^k, st_s, st_{\hat{A}}, \epsilon)$, where the values of the state st_s of the emulated session s are as follows, $s_{actor} = \hat{A}$, $s_{peer} = \hat{B}$, $s_{role} = \mathcal{I}$, $s_{sent} = \epsilon$, $s_{recv} = \epsilon$, $s_{status} = \mathtt{active}$, $s_{key} = \perp$, $s_{rand} = z$ (with $z \in_R \{0,1\}^k$), $s_{data} = \perp$, and $s_{step} = 1$. The algorithm Ψ outputs a message m, an updated session state, and an updated state of the user memory. The adversary then creates an initiator session s_1 of user $\hat{A}$ with peer $\hat{B}$ via the query $\mathsf{create}(\hat{A}, \mathcal{I}, \hat{B})$. The latter query returns a message m_1. He then creates a responder session s_2 of user $\hat{B}$ via the query $\mathsf{create}(\hat{B}, \mathcal{R}, \hat{A})$, and sends the message m, previously obtained by executing Ψ, to session s_2 via the query $\mathsf{send}(\hat{B}, 1, m)$. As a response, the adversary receives the message m_2, which he sends to session s_1 via a send query. Upon receiving message m_2 in session s_1, $\hat{A}$ executes $\Psi(1^k, st_{s_1}, st_{\hat{A}}, m_2)$ and thereby completes the session. E now chooses the completed session s_1 as the test session, and reveals the long-term secret key of the peer of the test session, namely user $\hat{B}$, as well as the randomness of session s_2. This enables him to compute the session key of the test session by executing Ψ on input $(1^k, st_{\tilde{s}}, st_{\hat{B}}, m_1)$, where m_1 is the outgoing message of session s_1 and the values of the session state $st_{\tilde{s}}$ are as follows, $s_{actor} = \hat{B}$, $s_{peer} = \hat{A}$, $s_{role} = \mathcal{R}$, $s_{sent} = \epsilon$, $s_{recv} = \epsilon$, $s_{status} = \mathtt{active}$, $s_{key} = \perp$, s_{rand} is the randomness of session s_2, $s_{step} = 2$, and s_{data} is obtained by executing the first protocol step. E thereby emulates a matching session of user $\hat{B}$. The previous attack shows that protocol π is insecure in the $\Omega_{\mathsf{AKE}\cap\mathsf{SL}}$ model. Note that the test session is fresh in $\Omega_{\mathsf{AKE}\cap\mathsf{SL}}$ since there is no freshness condition on a session that is an origin-session for the test session but no partially matching session for the test session in the definition of $F_{\Omega_{\mathsf{AKE}\cap\mathsf{SL}}}$. Also, there is no matching session for the test session s_1 since the message m sent by the adversary to session s_2 is different, with overwhelming probability, from the message m_1 output by session s_1, by Definition 6. □

The notion of c-origin-session is a stronger notion than origin-session as the former additionally requires distinct roles and agreement on the communicating users.

Definition 14 (c-origin-session)**.** *We say that a session s' with $s'_{status} \neq \perp$ is a c-origin-session for a session s with $s_{status} =$* `accepted` *if $s_{actor} = s'_{peer} \wedge s_{peer} = s'_{actor} \wedge s_{recv} = s'_{sent} \wedge s_{role} \neq s'_{role}$.*

Note that for protocols in the class INDP the last condition of Definition 13 can be simplified. This follows from the fact that a created initiator session s of any protocol in the class INDP completes upon receipt of a valid message m and the variable $s_{recv} = \epsilon$ is then updated with message m, i. e. $s_{recv} \leftarrow (s_{recv}, m)$. Consequently, the point in time of receipt of the first message m in an initiator session s coincides with the completion of session s. The same reasoning is applicable to responder sessions. It follows from (a) the previous observation, (b) the model $\Omega_{\mathsf{AKE}\cap\mathsf{SL}}$, and (c) Proposition 1 that a strong model for analyzing protocols in the class $\mathsf{INDP}\cap\mathsf{SL}$ is model $\Omega_{\mathsf{INDP}\cap\mathsf{SL}}$ defined as follows.

Definition 15 ($\Omega_{\mathsf{INDP}\cap\mathsf{SL}}$)**.** *The $\Omega_{\mathsf{INDP}\cap\mathsf{SL}}$ model is defined by (Q, F), where $Q = Q_{\mathsf{noCR}}$, and a session s is said to satisfy F if all of the following conditions hold:*

1. *no* session-key(s) *query has been issued,*
2. *for all sessions s^* such that s^* matches s, no* session-key(s^*) *query has been issued,*
3. *not both queries* corrupt(s_{actor}) *and* randomness(s) *have been issued,*
4. *for all sessions s' such that s' is a c-origin-session for session s, not both queries* corrupt(s_{peer}) *and* randomness(s') *have been issued, and*
5. *if there exists no origin-session for session s, then no* corrupt(s_{peer}) *query has been issued before the completion of session s.*

It is not difficult to verify that protocol $\mathrm{SIG_t}$(NAXOS) obtained by applying a tagged version of the signature transformation SIG from [18] to the NAXOS protocol is secure in model $\Omega_{\mathsf{INDP}\cap\mathsf{SL}}$. The outgoing message of a $\mathrm{SIG_t}$(NAXOS) initiator session of user $\hat{A}$ is of the form $(X, Sign_{\hat{A}}(0, X, \hat{B}))$ while the outgoing message of a $\mathrm{SIG_t}$(NAXOS) responder session of user $\hat{B}$ is of the form $(Y, Sign_{\hat{B}}(1, Y, \hat{A}))$.

We next show that any protocol in the class INDP-DH is insecure in $\Omega_{\mathsf{INDP}\cap\mathsf{SL}}$.

Proposition 2 (Impossibility result for INDP-DH)**.** *No one-round Diffie-Hellman type protocol in the class* INDP-DH *can satisfy security in the model $\Omega_{\mathsf{INDP}\cap\mathsf{SL}}$.*

Proof. Let π be an arbitrary protocol in INDP-DH. There exists an adversary E who wins the $W(\Omega_{\mathsf{INDP}\cap\mathsf{SL}})$ experiment against the challenger with non-negligible probability as follows. The adversary E first creates an initiator session s at $\hat{A}$ with peer $\hat{B}$ via the query create$(\hat{A}, \mathcal{I}, \hat{B})$ and receives as a response the message $m = (\delta, X)$, where X is the Diffie-Hellman exponential generated in session s and δ denotes optional public information. E chooses a value $z \in_R \mathbb{Z}_q$, computes $Z = g^z$, and sends message $\tilde{m} = (\delta', Z)$ to session s. Upon receiving message $\tilde{m}$ in session s, $\hat{A}$ executes $\Psi(1^k, st_s, st_{\hat{A}}, \tilde{m})$. E then chooses the completed session s as the test session and reveals the long-term secret key of user $\hat{B}$ via the query corrupt$(\hat{B})$. This enables him to compute the session key of the test session by executing Ψ on input $(1^k, st_{\tilde{s}}, st_{\hat{B}}, m)$, where $st_{\tilde{s}}$ is the state of the emulated matching session $\tilde{s}$. Note that the query public-info returned the initial values of additional variables stored in the user memory. This attack shows that π is insecure in $\Omega_{\mathsf{INDP}\cap\mathsf{SL}}$. It is a generalized version of the attack described by Krawczyk in [22, p. 15]. □

Remark 4. *Any protocol $\pi \in$ INDP-DH $\cap$ SL that does not contain sufficient public information in the outgoing message is insecure in the model derived from model $\Omega_{\mathsf{INDP}\cap\mathsf{SL}}$ and Proposition 2. This follows from the fact that a redirect event of a message from a session of a different user than the test session's peer can cause the existence of an origin-session that is not a c-origin-session for the test session. However, we do not distinguish between the relative strength of AKE protocols based on public information within the outgoing messages; adding such information does not provide additional security guarantees as it can be altered by the adversary.*

From the previous remark and Proposition 2 we derive the following security model.

Definition 16 ($\Omega_{\mathsf{INDP\text{-}DH}\cap\mathsf{SL}}$)**.** *The $\Omega_{\mathsf{INDP\text{-}DH}\cap\mathsf{SL}}$ model is defined by (Q, F), where $Q = Q_{\mathsf{noCR}}$ and a session s is said to satisfy F if all of the following conditions hold:*

1. *no* $\mathsf{session\text{-}key}(s)$ *query has been issued, and*
2. *for all sessions s^* such that s^* matches s, no* $\mathsf{session\text{-}key}(s^*)$ *query has been issued, and*
3. *not both queries* $\mathsf{corrupt}(s_{actor})$ *and* $\mathsf{randomness}(s)$ *have been issued, and*
4. *for all sessions s' such that s' is an origin-session for session s, not both queries* $\mathsf{corrupt}(s_{peer})$ *and* $\mathsf{randomness}(s')$ *have been issued, and*
5. *if there exists no origin-session for session s, then no* $\mathsf{corrupt}(s_{peer})$ *query has been issued.*

The model $\Omega_{\mathsf{INDP\text{-}DH}\cap\mathsf{SL}}$ is very similar to the eCK^w model defined in [18]. The NAXOS protocol [24] provides an example of a protocol in the class $\mathsf{INDP\text{-}DH} \cap \mathsf{SL}$ that is secure in the eCK^w model.

5 Models capturing chosen-randomness attacks

5.1 Deriving models with chosen-randomness

As an immediate consequence of Theorem 1, we obtain Theorem 2, which generalizes our impossibility results on protocol class $\mathsf{AKE} \cap \mathsf{SL}$ to adversaries who are in addition given access to the query $\mathsf{cr\text{-}create}$.

Theorem 2 (Impossibility result for $\mathsf{AKE} \cap \mathsf{SL}$ under chosen-randomness)**.** *Let π be an arbitrary protocol in the class $\mathsf{AKE} \cap \mathsf{SL}$. Let $X = (Q_{\mathsf{noCR}} \cup \{\mathsf{cr\text{-}create}\}, F)$ be the AKE security model with F being true for all sessions s and all sequences of queries. Let s^* denote the test session and let s' denote a session such that s^* is partially matching session s'. There exist adversaries who win the security experiment $W(X)$ against protocol π with non-negligible probability by issuing*

1. *a query* $\mathsf{session\text{-}key}(s^*)$*, or*
2. *a query* $\mathsf{session\text{-}key}(\tilde{s})$*, where $\tilde{s}$ and s^* are matching sessions, or*
3. *a query* $\mathsf{corrupt}(s^*_{actor})$ *and a (*$\mathsf{randomness}$ *or* $\mathsf{cr\text{-}create}$*) query to session s^*, or*
4. *a query* $\mathsf{corrupt}(s^*_{peer})$ *as well as a (*$\mathsf{randomness}$ *or* $\mathsf{cr\text{-}create}$*) query to session s', or*
5. *a query* $\mathsf{corrupt}(s^*_{peer})$ *before creation of session s^* via a (*create *or* $\mathsf{cr\text{-}create}$*) query or as long as $s^*_{status} = \mathtt{active}$ and $s^*_{recv} = \epsilon$, and impersonating the peer to the test session s^*.*

The previous theorem gives rise to the security model Ω^-_{AKE} defined as follows.

Definition 17 (Ω^-_{AKE})**.** *The Ω^-_{AKE} model is defined by (Q, F), where $Q = Q_{\mathsf{noCR}} \cup \{\mathsf{cr\text{-}create}\}$, and a session s is said to satisfy F if all of the following conditions hold:*

1. *no* $\mathsf{session\text{-}key}(s)$ *query has been issued,*
2. *for all sessions s^* such that s^* matches s, no* $\mathsf{session\text{-}key}(s^*)$ *query has been issued,*
3. *not both queries* $\mathsf{corrupt}(s_{actor})$ *and (*$\mathsf{randomness}(s)$ *or* $\mathsf{cr\text{-}create}(.)$ *creating session s) have been issued,*
4. *for all sessions s' such that s is partially matching session s', not both queries* $\mathsf{corrupt}(s_{peer})$ *and (*$\mathsf{randomness}(s')$ *or* $\mathsf{cr\text{-}create}(.)$ *creating session s') have been issued, and*
5. *if there exists no origin-session for session s, then no* $\mathsf{corrupt}(s_{peer})$ *query has been issued before creation of session s via a (*create *or* $\mathsf{cr\text{-}create}$*) query or as long as $s_{status} = \mathtt{active}$ and $s_{recv} = \epsilon$.*

The models Ω^-_{INDP} and $\Omega^-_{\mathsf{INDP\text{-}DH}}$, defined below, are obtained from the models $\Omega_{\mathsf{INDP}\cap\mathsf{SL}}$ and $\Omega_{\mathsf{INDP\text{-}DH}\cap\mathsf{SL}}$, respectively, in a similar way as model Ω^-_{AKE} is obtained from model $\Omega_{\mathsf{AKE}\cap\mathsf{SL}}$.

Definition 18 (Ω^-_{INDP})**.** *The Ω^-_{INDP} model is defined by (Q, F), where $Q = Q_{\mathsf{noCR}} \cup \{\mathsf{cr\text{-}create}\}$, and a session s is said to satisfy F if all of the following conditions hold:*

1. *no* $\mathsf{session\text{-}key}(s)$ *query has been issued,*

2. *for all sessions* s^* *such that* s^* *matches* s*, no* $\mathsf{session\text{-}key}(s^*)$ *query has been issued,*
3. *not both queries* $\mathsf{corrupt}(s_{actor})$ *and (*$\mathsf{randomness}(s)$ *or* $\mathsf{cr\text{-}create}(.)$ *creating session* s*) have been issued,*
4. *for all sessions* s' *such that* s' *is a c-origin-session for session* s*, not both queries* $\mathsf{corrupt}(s_{peer})$ *and (*$\mathsf{randomness}(s')$ *or* $\mathsf{cr\text{-}create}(.)$ *creating session* s'*) have been issued, and*
5. *if there exists no origin-session for session* s*, then no* $\mathsf{corrupt}(s_{peer})$ *query has been issued before the completion of session* s*.*

Definition 19 ($\Omega^{-}_{\mathsf{INDP\text{-}DH}}$)**.** *The* $\Omega^{-}_{\mathsf{INDP\text{-}DH}}$ *model is defined by* (Q, F)*, where* $Q = Q_{\mathsf{noCR}} \cup \{\mathsf{cr\text{-}create}\}$ *and a session* s *is said to satisfy* F *if all of the following conditions hold:*

1. *no* $\mathsf{session\text{-}key}(s)$ *query has been issued, and*
2. *for all sessions* s^* *such that* s^* *matches* s*, no* $\mathsf{session\text{-}key}(s^*)$ *query has been issued, and*
3. *not both queries* $\mathsf{corrupt}(s_{actor})$ *and (*$\mathsf{randomness}(s)$ *or* $\mathsf{cr\text{-}create}(.)$ *creating session* s*) have been issued, and*
4. *for all sessions* s' *such that* s' *is an origin-session for session* s*, not both queries* $\mathsf{corrupt}(s_{peer})$ *and (*$\mathsf{randomness}(s')$ *or* $\mathsf{cr\text{-}create}(.)$ *creating session* s'*) have been issued, and*
5. *if there exists no origin-session for session* s*, then no* $\mathsf{corrupt}(s_{peer})$ *query has been issued.*

We show in Section 5.3 that security in our extended models, which additionally capture attacks based on chosen randomness, implies security against attacks exploiting repeated randomness failures.

5.2 Insecurity of stateless protocols against chosen-randomness attacks

Even though the following proposition states that no stateless protocol is secure in model $\Omega^{-}_{\mathsf{INDP\text{-}DH}}$, we show in Section 7 that stateful protocols can in fact achieve these stronger security guarantees.

Proposition 3 (Impossibility result for $\mathsf{AKE} \cap \mathsf{SL}$ under chosen-randomness)**.** *No protocol in the class* $\mathsf{AKE} \cap \mathsf{SL}$ *can satisfy security in the model* $\Omega^{-}_{\mathsf{INDP\text{-}DH}}$*.*

Proof. Let π be an arbitrary protocol in $\mathsf{AKE} \cap \mathsf{SL}$. There exists an adversary E that wins the $\Omega^{-}_{\mathsf{INDP\text{-}DH}}$ game against the challenger with non-negligible probability as follows. The adversary E first completes a regular execution between two users $\hat{A}$ and $\hat{B}$, i.e. $\hat{A}$ and $\hat{B}$ complete matching sessions s and s', respectively. He then issues a $\mathsf{randomness}$ query against the responder session s' of user $\hat{B}$. E creates another responder session s'' of user $\hat{B}$ via the query $\mathsf{cr\text{-}create}(\hat{B}, \mathcal{R}, str, \hat{A})$, where str denotes the randomness that he revealed from session s', and replays the messages from session s to session s''. As the randomness used in session s'' is identical to the randomness used in session s' and $\pi \in \mathsf{SL}$, the messages that E receives from session s'' are the same as the messages sent by session s'. Now, E chooses the completed session s' as the test session, and reveals the session key computed in session s'' via a $\mathsf{session\text{-}key}(s'')$ query. As the session keys computed in sessions s' and s'' are the same and both sessions are non-matching, the adversary learns the session key of the test session. Hence, the protocol π is insecure in the $\Omega^{-}_{\mathsf{INDP\text{-}DH}}$ model. A similar attack that involves $\mathsf{cr\text{-}create}$ queries on both sessions s' and s'' is sketched in [34, p. 119]. □

Corollary 1. *No protocol in the class* $\mathsf{AKE} \cap \mathsf{SL}$ *can satisfy security in either model* $\Omega^{-}_{\mathsf{INDP}}$ *or model* $\Omega^{-}_{\mathsf{AKE}}$*.*

Proof. By Proposition 3, we know that no protocol in the class $\mathsf{AKE} \cap \mathsf{SL}$ can satisfy security in the model $\Omega^{-}_{\mathsf{INDP\text{-}DH}}$. The corollary now follows from the fact that the models $\Omega^{-}_{\mathsf{INDP}}$ and $\Omega^{-}_{\mathsf{AKE}}$ are both at least as strong as model $\Omega^{-}_{\mathsf{INDP\text{-}DH}}$ (by Proposition 9). □

5.3 Repeated randomness failures

In this section we show that security in the $\Omega^{-}_{\mathsf{AKE}}$ model implies security against repeated randomness. To this end, we compare the relative strength of security between the model $\Omega^{-}_{\mathsf{AKE}}$ and a very similar model to $\Omega^{-}_{\mathsf{AKE}}$, where the adversary is not given access to the query cr-create, but to the query reset-create. The latter query allows the adversary to create a session that uses the same randomness as used in a previous session of the same user. Practically, this models a flawed RNG that produces the same value more than once.

The query reset-create creates a new session with the same randomness as used in a previous session of the same user.

- reset-create$(\hat{P}, r, i[, \hat{Q}])$. The query models the creation of a new session s, using the same randomness as in session $s' = (\hat{P}, i)$, for the user $\hat{P}$. The query requires that $\hat{P} \in \mathcal{P}, \hat{Q} \in \mathcal{P}, r \in \{\mathcal{I}, \mathcal{R}\}$, and that $s'_{status} \neq \perp$; otherwise, it returns $\perp$. Session variables are initialized as

$$(s_{actor}, s_{role}, s_{sent}, s_{recv}, s_{status}, s_{key}, s_{rand}, s_{step}) \leftarrow (\hat{P}, r, \epsilon, \epsilon, \texttt{active}, \perp, s'_{rand}, 1) \ .$$

  If the optional peer identifier $\hat{Q}$ is provided, the variable s_{peer} is set to $\hat{Q}$.
  The key exchange algorithm Ψ is executed on input $(1^k, st_s, st_{\hat{P}}, \epsilon)$. The algorithm returns a triple $(m', st'_s, st'_{\hat{P}})$. We set $st_s \leftarrow st'_s$ and $st_{\hat{P}} \leftarrow st'_{\hat{P}}$. The query returns m'.

Consider the security model $X_{\mathsf{AKE}} = (Q_{\mathsf{noCR}} \cup \{\mathsf{reset\text{-}create}\}, F)$, where a session $s = (\hat{P}, i)$ is said to satisfy F if all of the following conditions hold:

1. no session-key(s) query has been issued,
2. for all sessions s^* such that s^* matches s, no session-key(s^*) query has been issued,
3. not both queries corrupt$(\hat{P})$ and (randomness(s) or reset-create(.) creating session s) have been issued,
4. for all sessions s' such that s is partially matching session s', not both queries corrupt(s_{peer}) and (randomness(s') or reset-create(.) creating session s') have been issued, and
5. if there exists no origin-session for session s, then no corrupt(s_{peer}) query has been issued before creation of session s or as long as $s_{status} = \texttt{active}$ and $s_{recv} = \epsilon$.

The relative strength of security between security models, first discussed by Choo et al. [14], is formally defined by Cremers and Feltz [18] as follows. Let $secure(M, \pi)$ be a predicate that is true if and only if the protocol π is secure in security model M.

Definition 20. *Let Π be a class of AKE protocols. We say that a security model M' is* at least as strong as *a security model M with respect to Π, denoted by $M \leq^{\Pi}_{Sec} M'$, if*

$$\forall\ \pi \in \Pi.\ secure(M', \pi) \Rightarrow secure(M, \pi). \tag{1}$$

Proposition 4. *Let $\Pi = \mathsf{AKE}$. The model $\Omega^{-}_{\mathsf{AKE}}$ is at least as strong as the model X_{AKE} with respect to Π according to Definition 20.*

Proof. The first condition of Definition 5 is satisfied since matching is defined in the same way for both models $\Omega^{-}_{\mathsf{AKE}}$ and X_{AKE}. Let $\pi \in \Pi$. To show that the second and third condition of Definition 5 hold, we construct an adversary E' attacking protocol π in model $\Omega^{-}_{\mathsf{AKE}}$ using an adversary E attacking π in model X_{AKE}. Adversary E' proceeds as follows. Whenever E issues a query create, corrupt, randomness, session-key or test-session, adversary E' issues the same query and forwards the answer received to E. Whenever E issues a query reset-create$(\hat{P}, r, i[, \hat{Q}])$ to create a new session of user $\hat{P}$, adversary E' first checks whether the status of session $s = (\hat{P}, i)$ is different from $\perp$. If this is the case, then E' issues the following sequence of queries: 1. randomness$(\hat{P}, i)$, and 2. cr-create$(\hat{P}, r, s_{rand}[, \hat{Q}])$. At the end of E's execution, i. e. after it has output its guess bit b, E' outputs b as well. Note that if F holds for the test session, then the freshness condition of model $\Omega^{-}_{\mathsf{AKE}}$ is also satisfied. In particular, if there exists a partially

matching session for the test session and the actors of both sessions are the same, then there is no corrupt query on that user in case a query reset-create was issued to create either of the two sessions. If there exists a partially matching session for the test session and the actors of both sessions are different, then a reset-create query on the test session only involves a query cr-create and a query randomness on another session of the test session's actor; issuing a corrupt query on the test session's peer does not render the test session un-fresh in model $\Omega^{-}_{\mathsf{AKE}}$ as long as there is no query randomness on the partially matching session. Hence, it holds that $Adv^{\pi,E}_{W(X_{\mathsf{AKE}})}(k) \leq Adv^{\pi,E'}_{W(\Omega^{-}_{\mathsf{AKE}})}(k)$, where k denotes the security parameter. Since by assumption protocol π is secure in $\Omega^{-}_{\mathsf{AKE}}$, there is a negligible function g such that $Adv^{\pi,E'}_{W(\Omega^{-}_{\mathsf{AKE}})}(k) \leq g(k)$. It follows that protocol π is secure in X_{AKE}. □

We obtain the following corollary as an immediate consequence of Proposition 4.

Corollary 2. *Let* $\Pi = \mathsf{AKE}$. *The model* $\Omega^{-}_{\mathsf{AKE}}$ *is at least as strong as the model* $Y_{\mathsf{AKE}} = (Q_{\mathsf{R}} \cup \{\mathsf{corrupt}, \mathsf{session\text{-}key}, \mathsf{reset\text{-}create}\}, F')$ *with respect to* Π *according to Definition 20, where* F' *is obtained from predicate* F *above by removing the* randomness *query from the conditions.*

Proof. Since $X_{\mathsf{AKE}} \leq^{\Pi}_{Sec} \Omega^{-}_{\mathsf{AKE}}$ (by Proposition 4) and $Y_{\mathsf{AKE}} \leq^{\Pi}_{Sec} X_{\mathsf{AKE}}$ (by a similar reduction proof as in the proof of Proposition 4), it follows that $Y_{\mathsf{AKE}} \leq^{\Pi}_{Sec} \Omega^{-}_{\mathsf{AKE}}$ by transitivity of Implication (1). □

6 Impossibility results and strong models for stateful protocols

We next present a generalized impossibility result and its derived model for the broad class AKE taking into account the adversary's ability to choose session-specific randomness. Recall that the completion of a session occurs at the time at which the status of the session is set to `accepted`.

Corollary 3 (Impossibility result for AKE)**.** *Let* π *be an arbitrary protocol in the class* AKE. *Let* $X = (Q_{\mathsf{noCR}} \cup \{\mathsf{cr\text{-}create}\}, F)$ *be the AKE security model with* F *being true for all sessions* s *and all sequences of queries. Let* s^* *denote the test session and let* s' *denote a session such that* s^* *is partially matching session* s'. *There exist adversaries who win the security experiment* $W(X)$ *against protocol* π *with non-negligible probability by issuing either*

1. *a query* session-key(s^*), *or*
2. *a query* session-key$(\tilde{s})$, *where* $\tilde{s}$ *and* s^* *are matching sessions, or*
3. *a query* corrupt(s^*_{actor}) *as well as (*randomness *or* cr-create*) queries on all sessions* s *with* $s_{actor} = s^*_{actor}$, *where the query* create *or* cr-create *creating session* s *occurred before completion of session* s^*, *or*
4. *a query* corrupt(s^*_{peer}) *as well as (*randomness *or* cr-create*) queries on all sessions* s *with* $s_{actor} = s'_{actor}$, *where the query* create *or* cr-create *creating session* s *occurred before completion of session* s', *or*
5. *a query* corrupt(s^*_{peer}) *before creation of session* s^* *via a query (*create *or* cr-create*) or as long as* $s^*_{status} =$ `active` *and* $s^*_{recv} = \epsilon$, *and impersonating the peer to the test session* s^*.

Proof. Corollary 3 follows from Theorem 1 and the following observation. Given the initial value of all the variables stored in the user memory of user $\hat{P}$ returned as response to the query public-info, the randomness used in all sessions of user $\hat{P}$ that are created before completion of the test session, and the long-term secret key of $\hat{P}$, the adversary can emulate the protocol execution steps on behalf of user $\hat{P}$ by executing Ψ sequentially to recompute the session-specific and user-specific data from the first session of user $\hat{P}$ to the relevant session of user $\hat{P}$. □

Remark 5 (On Corollary 3)**.** *Consider a protocol* $\pi \in$ AKE *storing previously seen messages to prevent replay attacks. Then, in case a session* s *receives a message* m *sent by some session*

s', the user memory of user s_{actor} gets updated after receiving this message. Thus, if another session s'' of user s_{actor}, created before s was created, is activated via a query send *with the same message m later on, then session s'' is aborted as it detected a previously received message (even if the session was created before session s). Similar situations might occur if updates of the user state occurring in later steps of the protocol execution involve the randomness of the session. Therefore, some of the attacks in Corollary 3 require the randomness of all sessions created prior to completion of the target session or its partially matching session.*

Corollary 3 gives rise to the model Ω_{AKE} defined as follows.

Definition 21 (Ω_{AKE})**.** *The model Ω_{AKE} is defined by (Q, F), where $Q = Q_{\mathsf{noCR}} \cup \{\mathsf{cr\text{-}create}\}$ and a session s is said to satisfy F if all of the following conditions hold:*

1. *no* session-key(s) *has been issued,*
2. *for all sessions s^* such that s^* matches s, no* session-key(s^*) *query has been issued,*
3. *not all queries* corrupt(s_{actor}) *as well as (*randomness *or* cr-create*) queries on all sessions $\tilde{s}$ with $\tilde{s}_{actor} = s_{actor}$, where the query* create *or* cr-create *creating session $\tilde{s}$ occurred before completion of session s, have been issued,*
4. *for all sessions s' such that s is partially matching session s', not all queries* corrupt(s_{peer}) *as well as (*randomness *or* cr-create*) on all sessions $\tilde{s}$ with $\tilde{s}_{actor} = s'_{actor}$, where the query* create *or* cr-create *creating session $\tilde{s}$ occurred before completion of session s', have been issued, and*
5. *if there exists no origin-session for session s, then no* corrupt(s_{peer}) *query has been issued before creation of session s via a query (*create *or* cr-create*) or as long as $s_{status} =$* `active` *and $s_{recv} = \epsilon$.*

In this work we restrict ourselves to the subclass ISM (Initial State Modification) of the class AKE and provide strong models for analyzing protocols in this subclass. The class ISM contains all AKE protocols that may *only* access and update user memory upon creation of sessions. It contains, e. g., the CNX protocol as well as the NXPR protocol, which we present in Section 7.[1] We leave the analysis of protocols that update user memory at later steps in the protocol execution as future work.

From the definition of the class ISM and from Corollary 3, we derive the following model.

Definition 22 ($\Omega_{\mathsf{AKE}\cap\mathsf{ISM}}$)**.** *The model $\Omega_{\mathsf{AKE}\cap\mathsf{ISM}}$ is defined by (Q, F), where $Q = Q_{\mathsf{noCR}} \cup \{\mathsf{cr\text{-}create}\}$ and a session s is said to satisfy F if all of the following conditions hold:*

1. *no* session-key(s) *has been issued,*
2. *for all sessions s^* such that s^* matches s, no* session-key(s^*) *query has been issued,*
3. *not all queries* corrupt(s_{actor}) *as well as (*randomness *or* cr-create*) on all sessions $\tilde{s}$ with $\tilde{s}_{actor} = s_{actor}$, where the query* create *or* cr-create *creating session $\tilde{s}$ occurred before or at creation of session s, have been issued,*
4. *for all sessions s' such that s is partially matching session s', not all queries* corrupt(s_{peer}) *as well as (*randomness *or* cr-create*) on all sessions $\tilde{s}$ with $\tilde{s}_{actor} = s'_{actor}$, where the query* create *or* cr-create *creating session $\tilde{s}$ occurred before or at creation of session s', have been issued, and*
5. *if there exists no origin-session for session s, then no* corrupt(s_{peer}) *query has been issued before creation of session s via a query (*create *or* cr-create*) or as long as $s_{status} =$* `active` *and $s_{recv} = \epsilon$.*

Whereas in the models that we defined in Section 4 and Section 5, the adversary is not allowed to compromise both the randomness of the target session and the long-term secret key of the actor of the target session, the adversary is allowed to compromise the latter values in the

[1] Note that this subclass does not contain protocols providing message replay detection as such protocols need to access and update the list of received messages stored in the user memory upon receipt of a message. However the subclass contains protocols such as NAXOS and CMQV, which do not modify the user memory.

$\Omega_{\mathsf{AKE}\cap\mathsf{ISM}}$ model and its descendants as long as the randomness of at least one of the previous sessions of the actor of the target session has not been compromised.

Definition 22 and Proposition 1 give rise to the model $\Omega_{\mathsf{INDP}\cap\mathsf{ISM}}$ defined below. The third condition in Definition 23 prevents the adversary from both corrupting the actor of the target session and revealing the randomness of all sessions of this user that were created prior to creation of the target session as well as the randomness of the target session itself. As the user memory is accessed and updated upon creation of sessions only, the responses for the previous queries would allow the adversary to emulate the protocol execution steps for the target session. The fourth condition specifies a similar requirement for sessions that are c-origin-sessions for the target session.

Definition 23 ($\Omega_{\mathsf{INDP}\cap\mathsf{ISM}}$)**.** *The model $\Omega_{\mathsf{INDP}\cap\mathsf{ISM}}$ is defined by (Q, F), where $Q = Q_{\mathsf{noCR}} \cup \{\mathsf{cr\text{-}create}\}$ and a session s is said to satisfy F if all of the following conditions hold:*

1. *no* $\mathsf{session\text{-}key}(s)$ *has been issued,*
2. *for all sessions s^* such that s^* matches s, no* $\mathsf{session\text{-}key}(s^*)$ *query has been issued,*
3. *not all queries* $\mathsf{corrupt}(s_{actor})$ *as well as (*$\mathsf{randomness}$ *or* $\mathsf{cr\text{-}create}$*) on all sessions $\tilde{s}$ with $\tilde{s}_{actor} = s_{actor}$, where the query* create *or* $\mathsf{cr\text{-}create}$ *creating session $\tilde{s}$ occurred before or at creation of session s, have been issued,*
4. *for all sessions s' such that s' is a c-origin-session for session s, not all queries* $\mathsf{corrupt}(s_{peer})$ *as well as (*$\mathsf{randomness}$ *or* $\mathsf{cr\text{-}create}$*) on all sessions $\tilde{s}$ with $\tilde{s}_{actor} = s'_{actor}$, where the query* create *or* $\mathsf{cr\text{-}create}$ *creating session $\tilde{s}$ occurred before or at creation of session s', have been issued, and*
5. *if there exists no origin-session for session s, then no* $\mathsf{corrupt}(s_{peer})$ *query has been issued before the completion of session s.*

Definition 22 and Propositions 1 and 2 give rise to the model $\Omega_{\mathsf{INDP\text{-}DH}\cap\mathsf{ISM}}$ defined as follows.

Definition 24 ($\Omega_{\mathsf{INDP\text{-}DH}\cap\mathsf{ISM}}$)**.** *The model $\Omega_{\mathsf{INDP\text{-}DH}\cap\mathsf{ISM}}$ is defined by (Q, F), where $Q = Q_{\mathsf{noCR}} \cup \{\mathsf{cr\text{-}create}\}$ and a session s is said to satisfy F if all of the following conditions hold:*

1. *no* $\mathsf{session\text{-}key}(s)$ *has been issued,*
2. *for all sessions s^* such that s^* matches s, no* $\mathsf{session\text{-}key}(s^*)$ *query has been issued,*
3. *not all queries* $\mathsf{corrupt}(s_{actor})$ *as well as (*$\mathsf{randomness}$ *or* $\mathsf{cr\text{-}create}$*) on all sessions $\tilde{s}$ with $\tilde{s}_{actor} = s_{actor}$, where the query* create *or* $\mathsf{cr\text{-}create}$ *creating session $\tilde{s}$ occurred before or at creation of session s, have been issued,*
4. *for all sessions s' such that s' is an origin-session for session s, not all queries* $\mathsf{corrupt}(s_{peer})$ *as well as (*$\mathsf{randomness}$ *or* $\mathsf{cr\text{-}create}$*) on all sessions $\tilde{s}$ with $\tilde{s}_{actor} = s'_{actor}$, where the query* create *or* $\mathsf{cr\text{-}create}$ *creating session $\tilde{s}$ occurred before or at creation of session s', have been issued, and*
5. *if there exists no origin-session for session s, then no* $\mathsf{corrupt}(s_{peer})$ *query has been issued.*

We formally study the relations between the different security models that we derived from impossibility results in Section 8.

Remark 6 (comparison with [10])**.** *Boyd and González Nieto [10] show that one-round AKE protocols that do not provide message replay detection cannot achieve PFS if the adversary can reveal session-specific randomness of the target session's peer. The attack on which their argument is based is disallowed in model $\Omega_{\mathsf{INDP}\cap\mathsf{ISM}}$, but only if the replayed message originates from the first created session of the target session's peer. In case the target session receives a message replayed from the n'th session of its peer, where $n > 1$, then the adversary is allowed to compromise the randomness of the n'th session of the peer as well as the long-term secret key of the peer after the end of the target session as long as he does not reveal the randomness of the previous $n - 1$ sessions of the peer. Our results thus reflect the impossibility result of Boyd and González Nieto on the class of protocols* ISM *only if the replayed message originates from the*

first session of the target session's peer. In particular, the attack in the proof of Theorem 1 from which we derived the fourth condition in Definition 13 can be seen as a generalized version of Boyd and González Nieto's attack.

7 Construction of strongly secure stateful protocols

In the previous sections we derived strong models capturing chosen-randomness attacks from impossibility results on given protocol classes. We have seen that stateless protocols fail to achieve security in these models. In this section we provide stateful variants of the NAXOS protocol [24], which are secure against attacks based on chosen randomness.

7.1 Protocol CNX

The CNX protocol ("Counter-NaXos"), shown in Figure 2, is a variant of the NAXOS protocol [24] and provides an example of a protocol from the class $\mathsf{INDP\text{-}DH} \setminus \mathsf{SL}$, where $H_1 : \{0,1\}^* \to \mathbb{Z}_p$ and $H_2 : \{0,1\}^* \to \{0,1\}^k$ denote two hash functions. In contrast to the NAXOS protocol, protocol CNX additionally includes a global counter value, which is shared across the sessions of a user, as input to the hash function H_1. We assume that each user maintains a counter l, taking values in $\mathbb{N}$, initialized with 0 and incremented by one upon creation of a new session. This counter variable is stored in the user memory. We write $st_{\hat{P}}.l$ to access the counter variable l of user $\hat{P}$. The CNX protocol proceeds as follows.

1. Upon creation of a new initiator session $s = (\hat{A}, i)$ with a $\mathsf{create}(\hat{A}, \mathcal{I}, \hat{B})$ query, user $\hat{A}$ increments the counter variable $st_{\hat{A}}.l \leftarrow st_{\hat{A}}.l + 1$, sets $s_{data} \leftarrow st_{\hat{A}}.l$, computes an outgoing public key $X \leftarrow g^{H_1(s_{rand}, a, s_{data})}$, and returns X as an outgoing message.
2. Upon creation of a new responder session $s' = (\hat{B}, j)$ with a $\mathsf{create}(\hat{B}, \mathcal{R}, \hat{A})$ query, user $\hat{B}$ increments the counter variable $st_{\hat{B}}.l \leftarrow st_{\hat{B}}.l + 1$ and sets $s'_{data} \leftarrow st_{\hat{B}}.l$.
3. When the responder receives message X via a $\mathsf{send}(\hat{B}, j, X)$ query, he computes an outgoing public key $Y \leftarrow g^{H_1(s'_{rand}, b, s'_{data})}$ and returns Y as an outgoing message. He computes a session key $s'_{key} \leftarrow H_2(A^{H_1(s'_{rand}, b, s'_{data})}, X^b, X^{H_1(s'_{rand}, b, s'_{data})}, \hat{A}, \hat{B})$ and accepts $s'_{status} \leftarrow \mathtt{accepted}$.
4. When the initiator receives message Y via a $\mathsf{send}(\hat{A}, i, Y)$ query, he computes a session key $s_{key} \leftarrow H_2(Y^a, B^{H_1(s_{rand}, a, s_{data})}, Y^{H_1(s_{rand}, a, s_{data})}, \hat{A}, \hat{B})$ and accepts $s_{status} \leftarrow \mathtt{accepted}$.

Remark 7. *In the specification of the* CNX *protocol given in Figure 2, the variable s_{data} stores the current value of the counter. The H_1 values, which depend on the value of the counter, are recomputed in the session key computation. In particular, it is essential for initiator sessions to store the counter value in the variable s_{data} to prevent the use of a different counter value in the session key computation due to other activations of sessions of the same user in between creation of the session and completion of the session. As an alternative specification of the* CNX *protocol, one could consider storing the exponent $H_1(s_{rand}, a, st_{\hat{A}}.l)$ in the variable s_{data}. Then users would not need to recompute the H_1 value in the session key computation.*

The following proposition states that the CNX protocol is secure in model $\Omega^{-}_{\mathsf{INDP\text{-}DH}}$.

Proposition 5. *Under the GAP-CDH assumption in the cyclic group G of prime order p, the* CNX *protocol is secure in model $\Omega^{-}_{\mathsf{INDP\text{-}DH}}$, when H_1, H_2 are modeled as independent random oracles.*

We refer the reader to Appendix A for the proof of Proposition 5.

By a straightforward adaptation of the proof of [18, Theorem 1] via integration of the $\mathsf{cr\text{-}create}$ query into the security models, we can show that protocol $\mathrm{SIG_t}$(CNX) obtained by applying the tagged version of the signature transformation SIG suggested in Section 4 to the CNX protocol is

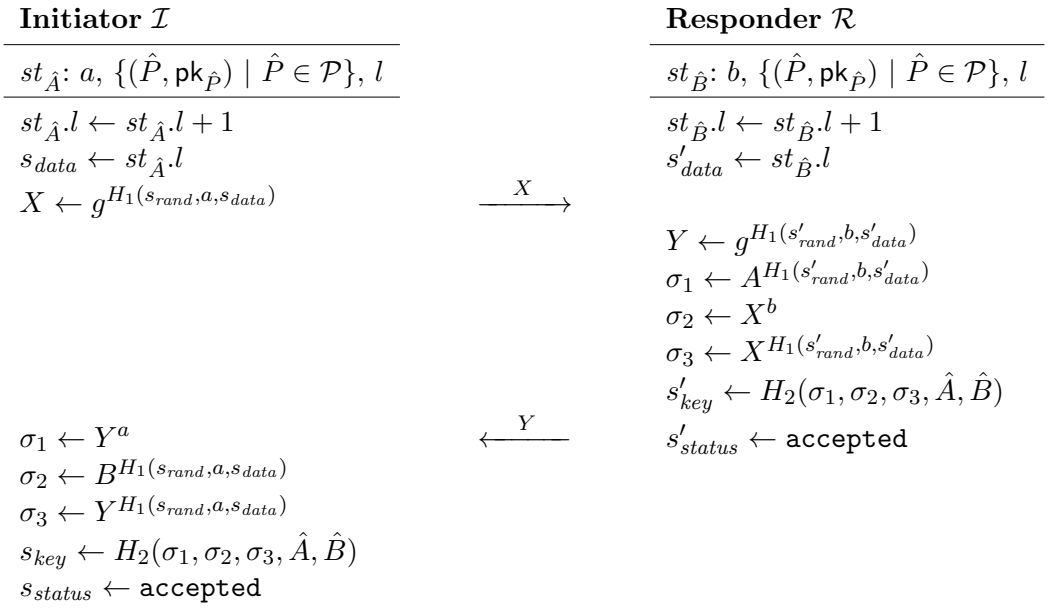

Figure 2: CNX protocol

secure in model $\Omega^{-}_{\mathsf{INDP}}$. Similarly, we can show that protocol SIG*(CNX), obtained by applying the signature transformation SIG with optional fields to the CNX protocol, is secure in model $\Omega^{-}_{\mathsf{AKE}}$.

Remark 8 (sequence numbers)**.** *In contrast to the use of sequence numbers to uniquely identify messages, the recipient of a message during execution of the* CNX *protocol does not need to store and maintain state information of each possible verifier to detect previously used sequence numbers (see also [27, p. 399]). However, as stated in [10], protocols using sequence numbers to order messages are secure against Boyd and González Nieto's replay attack [10, p. 458]. Moreover such protocols are secure against reset-and-replay attacks. Thus, designers of protocols face a trade-off between efficiency and security against various types of replay attacks.*

Remark 9 (comparison with [34])**.** *Yang et al. [34] argue that whenever the randomness of one session is identical to the randomness of another session of the same user, the adversary can learn the session key of either of the two sessions by performing a replay attack combined with a* session-key *query (as both sessions compute the same session key, but are non-matching). While Proposition 3 confirms that this statement holds for all protocols in the class* $\mathsf{AKE} \cap \mathsf{SL}$*, we have shown that there exists a protocol in* AKE*, namely* CNX*, that achieves security even under such reset-and-replay attacks against the target session.*

7.2 Protocol NXPR

Even though the CNX protocol is secure in model $\Omega^{-}_{\mathsf{INDP\text{-}DH}}$, it fails to achieve security in the stronger model $\Omega_{\mathsf{INDP\text{-}DH} \cap \mathsf{ISM}}$, as the following proposition shows.

Proposition 6. *The* CNX *protocol is insecure in model* $\Omega_{\mathsf{INDP\text{-}DH} \cap \mathsf{ISM}}$.

Proof. The following attack shows that the CNX protocol is insecure in $\Omega_{\mathsf{INDP\text{-}DH} \cap \mathsf{ISM}}$. The adversary creates an initiator session s of user $\hat{A}$ via the query $\mathsf{create}(\hat{A}, \mathcal{I}, \hat{B})$ and an initiator session of user $\hat{B}$ by issuing the query $\mathsf{create}(\hat{B}, \mathcal{I}, \hat{C})$. He then creates a responder session s' via the query $\mathsf{create}(\hat{B}, \mathcal{R}, \hat{A})$ and activates session s' by sending the message $X = g^x$ sent by session s to session s'. The adversary then sends message Y sent by session s' to session s. Session s accepts the key $s_{key} = H_2(Y^a, B^{H_1(s_{rand}, a, s_{data})}, Y^{H_1(s_{rand}, a, s_{data})}, \hat{A}, \hat{B})$ as the session key, while session s' accepts as its key $s'_{key} = H_2(A^{H_1(s'_{rand}, b, s'_{data})}, X^b, X^{H_1(s'_{rand}, b, s'_{data})}, \hat{A}, \hat{B})$. The

completed session s is chosen as the test session. Now a $\mathsf{randomness}$ query to session s' revealing the randomness of session s' followed by a $\mathsf{corrupt}(\hat{B})$ query revealing the long-term secret key of user $\hat{B}$, allows the adversary to compute the session key of the test session s (as he knows the counter value used in session s'). Note that the test session is fresh in $\Omega_{\mathsf{INDP\text{-}DH} \cap \mathsf{ISM}}$ since the adversary did not issue the query $\mathsf{randomness}$ or $\mathsf{cr\text{-}create}$ to the first created session of user $\hat{B}$. □

The NXPR protocol ("NaXos with Previous Randomness"), shown in Figure 3, is a variant of the NAXOS protocol [24] and provides an example of a protocol from the class $\mathsf{INDP\text{-}DH} \cap \mathsf{ISM}$, where $H_1 : \{0,1\}^* \to \mathbb{Z}_p$ and $H_2 : \{0,1\}^* \to \{0,1\}^k$ denote two hash functions. In contrast to the NAXOS protocol, protocol NXPR additionally includes the randomness of all sessions that have been previously created as input to the hash function H_1. We assume that each user maintains a variable $l \in \{0,1\}^*$ initialized with the empty string ϵ. We write $st_{\hat{P}}.l$ to access the variable l stored in the user memory of user $\hat{P}$.

Initiator $\mathcal{I}$		**Responder $\mathcal{R}$**
$st_{\hat{A}}$: a, $\{(\hat{P}, \mathsf{pk}_{\hat{P}}) \mid \hat{P} \in \mathcal{P}\}$, l		$st_{\hat{B}}$: b, $\{(\hat{P}, \mathsf{pk}_{\hat{P}}) \mid \hat{P} \in \mathcal{P}\}$, l
$s_{rand} \leftarrow \{0,1\}^k$		$s'_{rand} \leftarrow \{0,1\}^k$
$s_{data} \leftarrow st_{\hat{A}}.l$		$s'_{data} \leftarrow st_{\hat{B}}.l$
$st_{\hat{A}}.l \leftarrow (s_{rand}, s_{data})$		$st_{\hat{B}}.l \leftarrow (s'_{rand}, s'_{data})$
$X \leftarrow g^{H_1(s_{rand}, s_{data}, a)}$		
	$\xrightarrow{X}$	
		$Y \leftarrow g^{H_1(s'_{rand}, s'_{data}, b)}$
		$\sigma_1 \leftarrow A^{H_1(s'_{rand}, s'_{data}, b)}$
		$\sigma_2 \leftarrow X^b$
		$\sigma_3 \leftarrow X^{H_1(s'_{rand}, s'_{data}, b)}$
		$s'_{key} \leftarrow H_2(\sigma_1, \sigma_2, \sigma_3, \hat{A}, \hat{B})$
		$s'_{status} \leftarrow \mathtt{accepted}$
	$\xleftarrow{Y}$	
$\sigma_1 \leftarrow Y^a$		
$\sigma_2 \leftarrow B^{H_1(s_{rand}, s_{data}, a)}$		
$\sigma_3 \leftarrow Y^{H_1(s_{rand}, s_{data}, a)}$		
$s_{key} \leftarrow H_2(\sigma_1, \sigma_2, \sigma_3, \hat{A}, \hat{B})$		
$s_{status} \leftarrow \mathtt{accepted}$		

Figure 3: NXPR Protocol

Proposition 7. *Under the GAP-CDH assumption in the cyclic group G of prime order p, the* NXPR *protocol is secure in the $\Omega_{\mathsf{INDP\text{-}DH} \cap \mathsf{ISM}}$ model, when H_1, H_2 are modeled as independent random oracles.*

We refer the reader to Appendix B for the proof of Proposition 7.

By a straightforward adaptation of the proof of [18, Theorem 1] via integration of the $\mathsf{cr\text{-}create}$ query into the security models, we can show that protocol $\mathrm{SIG_t}$(NXPR) obtained by applying the tagged version of the signature transformation SIG suggested in Section 4 to the NXPR protocol is secure in model $\Omega_{\mathsf{INDP} \cap \mathsf{ISM}}$. Similarly, we can show that protocol SIG^*(NXPR), obtained by

applying the signature transformation SIG with optional fields to the NXPR protocol, is secure in model $\Omega_{\mathsf{AKE}\cap\mathsf{ISM}}$.

Remark 10 (user state comparison of NXPR to CNX)**.** *In contrast to the* CNX *protocol, the* NXPR *protocol requires each user to store the concatenation of the randomness generated in all his sessions in the user memory. Thus, before the n'th session, where $n > 1$, of user $\hat{P}$ is created, the user memory $st_{\hat{P}}$ contains its long-term secret key, the long-term public key of all users $\hat{Q} \in \mathcal{P}$, and the concatenation of $n-1$ bit strings of length k corresponding to the randomness generated in all previous sessions of user $\hat{P}$.*

8 Relations between the security models

In this section we illustrate the relations between the different security models that we established.

Proposition 8. *Let* Π *be protocol class* AKE*.*

- *The model* $\Omega_{\mathsf{INDP}\cap\mathsf{SL}}$ *is stronger than the model* $\Omega_{\mathsf{INDP\text{-}DH}\cap\mathsf{SL}}$ *with respect to* Π *according to Definition 20.*
- *The model* $\Omega_{\mathsf{AKE}\cap\mathsf{SL}}$ *is stronger than the model* $\Omega_{\mathsf{INDP}\cap\mathsf{SL}}$ *with respect to* Π *according to Definition 20.*

Proof. We first show that $\Omega_{\mathsf{INDP\text{-}DH}\cap\mathsf{SL}} \leq_{Sec}^{\mathsf{AKE}} \Omega_{\mathsf{INDP}\cap\mathsf{SL}}$. The proof that $\Omega_{\mathsf{INDP}\cap\mathsf{SL}} \leq_{Sec}^{\mathsf{AKE}} \Omega_{\mathsf{AKE}\cap\mathsf{SL}}$ proceeds in a very similar way. The first condition of Definition 5 is satisfied as matching is defined in the same way for both models $\Omega_{\mathsf{INDP\text{-}DH}\cap\mathsf{SL}}$ and $\Omega_{\mathsf{INDP}\cap\mathsf{SL}}$. To see that the second condition of Definition 5 holds, it suffices to show that if there exists an adversary E such that the probability of event Multiple-Match$_{\pi,E}^{W(\Omega_{\mathsf{INDP\text{-}DH}\cap\mathsf{SL}})}(k)$ is non-negligible, then there exists an adversary E' such that the probability of event Multiple-Match$_{\pi,E}^{W(\Omega_{\mathsf{INDP}\cap\mathsf{SL}})}(k)$ is non-negligible. This is straightforward. Let $\pi \in \Pi$. To show that the third condition of Definition 5 holds, we construct an adversary E' attacking protocol π in model $\Omega_{\mathsf{INDP}\cap\mathsf{SL}}$ using an adversary E attacking π in $\Omega_{\mathsf{INDP\text{-}DH}\cap\mathsf{SL}}$. Adversary E' proceeds as follows. Whenever E issues a query $q \in Q_{\mathsf{noCR}} \cup \{\mathsf{test\text{-}session}\}$, adversary E' issues the same query and forwards the answer received to E. At the end of E's execution, i. e., after it has output its guess bit b, E' outputs b as well. Note that if the freshness condition of $\Omega_{\mathsf{INDP\text{-}DH}\cap\mathsf{SL}}$ holds for the test session, then by definition the freshness condition of $\Omega_{\mathsf{INDP}\cap\mathsf{SL}}$ also holds. First, if there is no origin-session, then the fifth condition of the freshness condition of model $\Omega_{\mathsf{INDP\text{-}DH}\cap\mathsf{SL}}$ requires that there is no corrupt query has been issued on the peer of the test session, which implies the fifth condition of the freshness condition of model $\Omega_{\mathsf{INDP}\cap\mathsf{SL}}$. Second, if there is an origin-session s for the test session, then it is required in model $\Omega_{\mathsf{INDP\text{-}DH}\cap\mathsf{SL}}$ that not both queries corrupt on the peer of the test session and randomness on session s have been issued. Now, if session s is a c-origin-session for the test session, then the fourth condition of the freshness definition of model $\Omega_{\mathsf{INDP}\cap\mathsf{SL}}$ is satisfied. However, if session s is not a c-origin-session for the test session, then freshness in the model $\Omega_{\mathsf{INDP}\cap\mathsf{SL}}$ is guaranteed as well. Even though both of the critical queries are allowed in the model $\Omega_{\mathsf{INDP}\cap\mathsf{SL}}$, they do not occur since otherwise freshness in the model $\Omega_{\mathsf{INDP\text{-}DH}\cap\mathsf{SL}}$ would be violated. Hence, it holds that $Adv_{W(X)}^{\pi,E}(k) \leq Adv_{W(X)}^{\pi,E'}(k)$, where k denotes the security parameter. Since by assumption protocol π is secure in model $\Omega_{\mathsf{INDP}\cap\mathsf{SL}}$, there is a negligible function g such that $Adv_{W(X)}^{\pi,E'}(k) \leq g(k)$. It follows that protocol π is secure in model $\Omega_{\mathsf{INDP\text{-}DH}\cap\mathsf{SL}}$.

The NAXOS protocol provides an example of a protocol that is secure in the model $\Omega_{\mathsf{INDP\text{-}DH}\cap\mathsf{SL}}$, but insecure in $\Omega_{\mathsf{INDP}\cap\mathsf{SL}}$. The protocol SIG$_{\mathrm{t}}$(NAXOS) introduced in Section 4 provides an example of a protocol that is secure in $\Omega_{\mathsf{INDP}\cap\mathsf{SL}}$, but insecure in $\Omega_{\mathsf{AKE}\cap\mathsf{SL}}$ as the following replay attack shows.

1. First the adversary establishes via a sequence of create and send queries an initiator session s of user $\hat{A}$ and a responder session s' of user $\hat{B}$ such that s and s' are matching sessions.

2. The adversary creates another initiator session s'' of user $\hat{A}$ with peer $\hat{B}$, and replays the message sent by session s' to session s''. Clearly, session s' is a c-origin-session for session s'', but s'' is not partially matching session s'.
3. He then chooses session s'' as the test session and issues the queries $\mathsf{corrupt}(\hat{B})$ and $\mathsf{randomness}(s')$.
4. Since the adversary knows the user state of user $\hat{B}$ and the randomness of session s', he can emulate the session key computation of a matching session and compute the session key of session s'' by executing Ψ on input $(1^k, st_{s'_p}, st_{\hat{B}}, s''_{sent})$, where $st_{s'_p}$ denotes the session-specific memory of session s' after its creation, but before its completion.

□

Proposition 9. *Let* Π *be protocol class* AKE.

- *The model* Ω^-_{INDP} *is stronger than the model* $\Omega^-_{\mathsf{INDP\text{-}DH}}$ *with respect to* Π *according to Definition 20.*
- *The model* Ω^-_{AKE} *is stronger than the model* Ω^-_{INDP} *with respect to* Π *according to Definition 20.*

Proof. The proofs that model Ω^-_{INDP} is at least as strong as the model $\Omega^-_{\mathsf{INDP\text{-}DH}}$ and that model Ω^-_{AKE} is at least as strong as the model Ω^-_{INDP} are similar to the proof of Proposition 8.

The CNX protocol provides an example of a protocol that is secure in model $\Omega^-_{\mathsf{INDP\text{-}DH}}$, but insecure in model Ω^-_{INDP} due to a PFS attack. The protocol $\mathrm{SIG_t}$(CNX) introduced in Section 7 provides an example of a protocol that is secure in model Ω^-_{INDP}, but insecure in model Ω^-_{AKE} due to the same replay attack as the one on the protocol $\mathrm{SIG_t}$(NAXOS) described in the proof of Proposition 8. □

Proposition 10. *Let* Π *be protocol class* AKE.

- *The model* $\Omega^-_{\mathsf{INDP\text{-}DH}}$ *is stronger than the model* $\Omega_{\mathsf{INDP\text{-}DH}\cap\mathsf{SL}}$ *with respect to* Π *according to Definition 20.*
- *The model* Ω^-_{INDP} *is stronger than the model* $\Omega_{\mathsf{INDP}\cap\mathsf{SL}}$ *with respect to* Π *according to Definition 20.*
- *The model* Ω^-_{AKE} *is stronger than the model* $\Omega_{\mathsf{AKE}\cap\mathsf{SL}}$ *with respect to* Π *according to Definition 20.*

Proof. The proof that model $\Omega^-_{\mathsf{INDP\text{-}DH}}$ is at least as strong as the model $\Omega_{\mathsf{INDP\text{-}DH}\cap\mathsf{SL}}$ proceeds in a similar way as the proof of Proposition 8. The same holds for the other pairs of models.

The protocols SIG^*(NAXOS), $\mathrm{SIG_t}$(NAXOS), and NAXOS provide examples of protocols that are secure in the models $\Omega_{\mathsf{AKE}\cap\mathsf{SL}}$, $\Omega_{\mathsf{INDP}\cap\mathsf{SL}}$, and $\Omega_{\mathsf{INDP\text{-}DH}\cap\mathsf{SL}}$, respectively. As all of these protocols are stateless, they fail to provide security in our models that give the adversary access to the query cr-create, by Proposition 3. □

Proposition 11. *Let* Π *be protocol class* AKE.

- *The model* $\Omega_{\mathsf{INDP}\cap\mathsf{ISM}}$ *is stronger than the model* $\Omega_{\mathsf{INDP\text{-}DH}\cap\mathsf{ISM}}$ *with respect to* Π *according to Definition 20.*
- *The model* $\Omega_{\mathsf{AKE}\cap\mathsf{ISM}}$ *is stronger than the model* $\Omega_{\mathsf{INDP}\cap\mathsf{ISM}}$ *with respect to* Π *according to Definition 20.*

Proof. The proof is similar to the proof of Proposition 8.

To prove that model $\Omega_{\mathsf{INDP}\cap\mathsf{ISM}}$ is stronger than model $\Omega_{\mathsf{INDP\text{-}DH}\cap\mathsf{ISM}}$, we need to show, among others, that the third condition of Definition 5 holds. Thus, we construct an adversary E' attacking protocol π in model $\Omega_{\mathsf{INDP}\cap\mathsf{ISM}}$ using an adversary E attacking π in model $\Omega_{\mathsf{INDP\text{-}DH}\cap\mathsf{ISM}}$, where $\pi \in \mathsf{AKE}$. Whenever E issues a query $q \in Q_{\mathsf{noCR}} \cup \{\mathsf{cr\text{-}create}, \mathsf{test\text{-}session}\}$, adversary E' issues the same query and forwards the answer received to E. Note that if the freshness condition of $\Omega_{\mathsf{INDP\text{-}DH}\cap\mathsf{ISM}}$ holds for the test session, then by definition the freshness condition of $\Omega_{\mathsf{INDP}\cap\mathsf{ISM}}$ also holds. First, if there is no origin-session, then the fifth condition of the freshness condition of model $\Omega_{\mathsf{INDP\text{-}DH}\cap\mathsf{ISM}}$ requires that there is no corrupt query has been issued on the peer of the

test session, which implies Condition 5 of the freshness condition of model $\Omega_{\mathsf{INDP}\cap\mathsf{ISM}}$. The case where there is an origin-session for the test session is treated is the same way as in the proof of Proposition 8. The proof that $\Omega_{\mathsf{INDP}\cap\mathsf{ISM}} \leq_{Sec}^{\mathsf{AKE}} \Omega_{\mathsf{AKE}\cap\mathsf{ISM}}$ proceeds in a very similar way.

The NXPR protocol presented in Section 7.2 provides an example of a protocol that is secure in the model $\Omega_{\mathsf{INDP\text{-}DH}\cap\mathsf{ISM}}$, but insecure in $\Omega_{\mathsf{INDP}\cap\mathsf{ISM}}$ due to a PFS attack. The protocol $\mathrm{SIG_t}$(NXPR) introduced in Section 7.2 provides an example of a protocol that is secure in model $\Omega_{\mathsf{INDP}\cap\mathsf{ISM}}$, but insecure in model $\Omega_{\mathsf{AKE}\cap\mathsf{ISM}}$ due to the same replay attack as the one on the protocol $\mathrm{SIG_t}$(NAXOS) described in the proof of Proposition 8.

□

Proposition 12. *Let* Π *be protocol class* AKE.

- *The model* $\Omega_{\mathsf{INDP\text{-}DH}\cap\mathsf{ISM}}$ *is stronger than the model* $\Omega^{-}_{\mathsf{INDP\text{-}DH}}$ *with respect to* Π *according to Definition 20.*
- *The model* $\Omega_{\mathsf{INDP}\cap\mathsf{ISM}}$ *is stronger than the model* $\Omega^{-}_{\mathsf{INDP}}$ *with respect to* Π *according to Definition 20.*
- *The model* $\Omega_{\mathsf{AKE}\cap\mathsf{ISM}}$ *is stronger than the model* $\Omega^{-}_{\mathsf{AKE}}$ *with respect to* Π *according to Definition 20.*

Proof. The proof that model $\Omega_{\mathsf{INDP\text{-}DH}\cap\mathsf{ISM}}$ is at least as strong as the model $\Omega^{-}_{\mathsf{INDP\text{-}DH}}$ proceeds in a similar way as the proof of Proposition 8. Among others, we need to show that the third condition of Definition 5 holds. Thus, we construct an adversary E' attacking protocol π in model $\Omega_{\mathsf{INDP\text{-}DH}\cap\mathsf{ISM}}$ using an adversary E attacking π in model $\Omega^{-}_{\mathsf{INDP\text{-}DH}}$, where $\pi \in \mathsf{AKE}$. Whenever E issues a query $q \in Q_{\mathsf{noCR}} \cup \{\mathsf{cr\text{-}create}, \mathsf{test\text{-}session}\}$, adversary E' issues the same query and forwards the answer received to E. Note that if the freshness condition of $\Omega^{-}_{\mathsf{INDP\text{-}DH}}$ is satisfied for the test session s, then the freshness condition of $\Omega_{\mathsf{INDP\text{-}DH}\cap\mathsf{ISM}}$ is also satisfied. The third condition of freshness in model $\Omega^{-}_{\mathsf{INDP\text{-}DH}}$ requires that not both queries $\mathsf{corrupt}(s_{actor})$ and ($\mathsf{randomness}(s)$ or $\mathsf{cr\text{-}create}(s)$) have been issued, which implies Conditions 3 of freshness in model $\Omega_{\mathsf{INDP\text{-}DH}\cap\mathsf{ISM}}$. The same argument applies to the fourth condition of the freshness definition.

The proofs that model $\Omega_{\mathsf{INDP}\cap\mathsf{ISM}}$ is at least as strong as model $\Omega^{-}_{\mathsf{INDP}}$ and that model $\Omega_{\mathsf{AKE}\cap\mathsf{ISM}}$ is at least as strong as model $\Omega^{-}_{\mathsf{AKE}}$ are similar to the proof of the previous statement.

The CNX protocol provides an example of a protocol that is secure in model $\Omega^{-}_{\mathsf{INDP\text{-}DH}}$, but insecure in model $\Omega_{\mathsf{INDP\text{-}DH}\cap\mathsf{ISM}}$ (see Proposition 6). The protocols $\mathrm{SIG_t}$(CNX) and $\mathrm{SIG^*}$(CNX) from Section 7.1 are secure in the models $\Omega^{-}_{\mathsf{INDP}}$ and $\Omega^{-}_{\mathsf{AKE}}$, respectively, but insecure in the corresponding lifted models $\Omega_{\mathsf{INDP}\cap\mathsf{ISM}}$ and $\Omega_{\mathsf{AKE}\cap\mathsf{ISM}}$ due to a similar attack as in the proof of Proposition 6. □

8.1 Protocol-security hierarchy

In Figure 4 we show the *protocol-security hierarchy* [2] for AKE security of the protocols developed in this paper with respect to security models of this paper. Each node in Figure 4 lists a protocol and the security model in which the protocol is secure. Arrows indicate stronger protocols, i. e., the protocol at the end of an arrow is not only secure in the same models as the protocol at the start of that arrow, but also in a stronger model, listed below the protocol name. Further, if there is an arrow between two nodes, then the protocol at the start of the arrow is insecure in the model indicated in the node at the end of the arrow. Thus, the protocol at the top of the hierarchy is the only one that is secure in all models in the figure. We discuss below the protocols of the hierarchy in Figure 4.

The rounded rectangles in Figure 4 identify protocol classes. For each class, we identify the security guarantees that cannot be achieved by any of its members. For example, because NAXOS is a member of $\mathsf{INDP\text{-}DH} \cap \mathsf{SL}$, it cannot provide PFS nor message origin authentication. Additionally, NAXOS is insecure if the RNG is compromised or fails (see Proposition 3). The

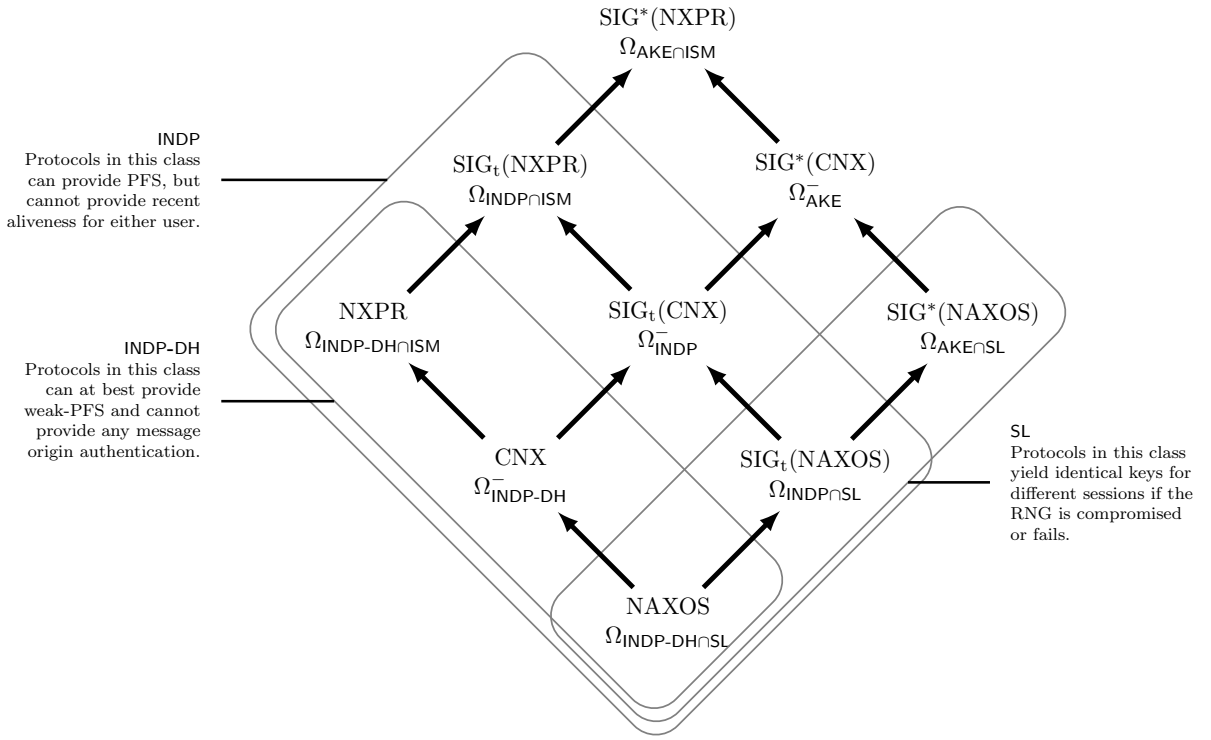

Figure 4: Annotated protocol-security hierarchy

protocols in the figure that belong to the class $\mathsf{AKE} \setminus \mathsf{SL}$ are secure even against attacks based on bad randomness such as reset-and-replay attacks. In contrast to the protocols CNX, SIG$_t$(CNX), and SIG*(CNX), the protocols NXPR, SIG$_t$(NXPR), and SIG*(NXPR), achieve security even under compromise of the randomness of the target session and the long-term secret key of the actor of that session as long as the randomness of at least one of the previous sessions of the same user has not been compromised.

The protocols in the class INDP can provide PFS, but cannot provide *recent aliveness*. Recent aliveness means that, after completion of a session with a certain peer, the user executing the session has a guarantee that its peer has been alive during the execution of the protocol [21]. For example, the protocol SIG$_t$(NAXOS) provides PFS as it is secure in the model $\Omega_{\mathsf{INDP}\cap\mathsf{SL}}$, but it does not provide recent aliveness, because the messages of initiator and responder can be generated independently of each other. In contrast, the protocols SIG*(NAXOS), SIG*(CNX), and SIG*(NXPR) provide recent aliveness to the initiator in the models $\Omega_{\mathsf{AKE}\cap\mathsf{SL}}$, Ω^-_{AKE}, and $\Omega_{\mathsf{AKE}\cap\mathsf{ISM}}$, respectively. The responder not only signs his own Diffie-Hellman exponential but also the exponential that he received from the initiator. Thus, the latter protocols also achieve security against replay attacks to the initiator. Recent aliveness for both initiator and responder can be achieved in three-message protocols, e. g., by adding a third message that contains a signature on the Diffie-Hellman exponential that the initiator received from his peer.

9 Conclusions

In this paper we clarified the limits of AKE security by giving formal impossibility results on the security of protocols that belong to different classes, and deriving security models for the respective protocol classes from these impossibility results. If a protocol designer aims to develop a protocol in a certain class, our results demonstrate which strong guarantees can be achieved by such protocols. Conversely, if a certain security guarantee is required, our results indicate in which protocol classes it can be achieved. For example, PFS under weak security assumptions on

the RNG can be achieved in the protocol class $\mathsf{AKE} \setminus (\mathsf{SL} \cup \mathsf{INDP\text{-}DH})$.

We provided new variants of the NAXOS protocol, which are secure against attacks based on chosen randomness; in these protocol variants, the state of a user's memory is modified during execution of the protocol. The CNX protocol is secure in the model $\Omega^{-}_{\mathsf{INDP\text{-}DH}}$. The NXPR protocol is secure in the stronger model $\Omega_{\mathsf{INDP\text{-}DH} \cap \mathsf{ISM}}$. Our results show that it is possible to construct secure AKE protocols under significantly weaker assumptions on the RNG than previously thought possible.

Our models do not cover adversarial registration of public keys [9]. Adding this capability would yield additional impossibility results and derived models, which we leave as future work.

References

[1] P. Abeni, L. Bello, and M. Bertacchini. Exploiting DSA-1571: How to break PFS in SSL with EDH. `http://www.lucianobello.com.ar/exploiting_DSA-1571/index.html` (Accessed 05/11/2013).

[2] D. Basin and C. Cremers. Degrees of security: Protocol guarantees in the face of compromising adversaries. In *Computer Science Logic, 24th International Workshop, CSL 2010, 19th Annual Conference of the EACSL*, volume 6247 of *LNCS*, pages 1–18. Springer, 2010.

[3] M. Bellare, Z. Brakerski, M. Naor, T. Ristenpart, G. Segev, H. Shacham, and S. Yilek. Hedged Public-Key encryption: How to protect against bad randomness. In *Advances in Cryptology - ASIACRYPT 2009*, LNCS, pages 232–249. Springer-Verlag, 2009.

[4] M. Bellare, D. Pointcheval, and P. Rogaway. Authenticated key exchange secure against dictionary attacks. In *19th International Conference on Theory and Application of Cryptographic Techniques*, EUROCRYPT'00, pages 139–155. Springer, 2000.

[5] M. Bellare and P. Rogaway. Entity authentication and key distribution. In *13th annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '93, pages 232–249. Springer New York, NY, USA, 1994.

[6] M. Bellare and P. Rogaway. Provably secure session key distribution: the three party case. In *27th annual ACM symposium on Theory of computing*, STOC '95, pages 57–66. ACM New York, NY, USA, 1995.

[7] S. Blake-Wilson, D. Johnson, and A. Menezes. Key agreement protocols and their security analysis. In M. Darnell, editor, *Crytography and Coding*, volume 1355 of *LNCS*, pages 30–45. Springer Berlin Heidelberg, 1997.

[8] S. Blake-Wilson and A. Menezes. Unknown key-share attacks on the Station-to-Station (STS) protocol. In I. H. and Z. Y., editors, *PKC '99 Proceedings of the Second International Workshop on Practice and Theory in Public Key Cryptography*, volume 1560 of *LNCS*, pages 154–170. Springer, 1999.

[9] C. Boyd, C. Cremers, M. Feltz, K. Paterson, B. Poettering, and D. Stebila. ASICS: Authenticated key exchange security incorporating certification systems. In J. Crampton, S. Jajodia, and K. Mayes, editors, *Computer Security – ESORICS 2013*, volume 8134 of *LNCS*, pages 381–399. Springer Berlin Heidelberg, 2013.

[10] C. Boyd and J. G. Nieto. On Forward Secrecy in One-Round Key Exchange. In *13th IMA International Conference, IMACC 2011*, volume 7089 of *LNCS*, pages 451–468. Springer, 2011.

[11] C. Brzuska, M. Fischlin, B. Warinschi, and S. Williams. Composability of bellare-rogaway key exchange protocols. In *Proceedings of the 18th ACM conference on Computer and communications security*, CCS '11, pages 51–62, New York, NY, USA, 2011. ACM.

[12] R. Canetti and H. Krawczyk. Analysis of key-exchange protocols and their use for building secure channels. In B. Pfitzmann, editor, *EUROCRYPT'01*, volume 2045 of *LNCS*, pages 453–474. Springer London, UK, 2001.

[13] Q. Cheng, C. Ma, and X. Hu. A new strongly secure authenticated key exchange protocol. In J. H. Park, H.-H. Chen, M. Atiquzzaman, C. Lee, T.-H. Kim, and S.-S. Yeo, editors, *ISA '09*, volume 5576 of *LNCS*, pages 135–144. Springer, 2009.

[14] K.-K. R. Choo, C. Boyd, and Y. Hitchcock. Examining indistinguishability-based proof models for key establishment protocols. In *Proceedings of the 11th international conference on Theory and Application of Cryptology and Information Security*, ASIACRYPT'05, pages 585–604, Berlin, Heidelberg, 2005. Springer-Verlag.

[15] C. Cremers. Examining indistinguishability-based security models for key exchange protocols: the case of CK, CK-HMQV, and eCK. In *Proceedings of the 6th ACM Symposium on Information, Computer and Communications Security*, ASIACCS '11, pages 80–91, New York, NY, USA, 2011. ACM.

[16] C. Cremers and M. Feltz. One-round strongly secure key exchange with perfect forward secrecy and deniability. Cryptology ePrint Archive, Report 2011/300, 2011. `http://eprint.iacr.org/`.

[17] C. Cremers and M. Feltz. Beyond eCK: Perfect Forward Secrecy under Actor Compromise and Ephemeral-Key Reveal. In *Proceedings of the 17th European conference on Research in computer security*, ESORICS, Berlin, Heidelberg, 2012. Springer-Verlag.

[18] C. Cremers and M. Feltz. Beyond eCK: Perfect forward secrecy under actor compromise and ephemeral-key reveal. *Designs, Codes and Cryptography*, 2013. `http://dx.doi.org/10.1007/s10623-013-9852-1`.

[19] F. Hao. On robust key agreement based on public key authentication. In *Financial Cryptography*, volume 6052 of *LNCS*, pages 383–390. Springer, 2010.

[20] M. Kim, A. Fujioka, and B. Ustaoglu. Strongly secure authenticated key exchange without naxos' approach. In *IWSEC'09*, pages 174–191, 2009.

[21] H. Krawczyk. SIGMA: The 'SIGn-and-MAc' Approach to Authenticated Diffie-Hellman and Its Use in the IKE-Protocols. In *CRYPTO*, pages 400–425, 2003.

[22] H. Krawczyk. HMQV: A high-performance secure Diffie-Hellman protocol. In V. Shoup, editor, *Advances in Cryptology - CRYPTO 2005*, volume 3621 of *LNCS*, pages 546–566. Springer, 2005.

[23] H. Krawczyk. HMQV: A High-Performance Secure Diffie-Hellman Protocol. Cryptology ePrint Archive, Report 2005/176, 2005. `http://eprint.iacr.org/`.

[24] B. LaMacchia, K. Lauter, and A. Mityagin. Stronger security of authenticated key exchange. In W. Susilo, J. K. Liu, and Y. Mu, editors, *ProvSec'07*, volume 4784 of *LNCS*, pages 1–16. Springer, 2007.

[25] J. Lee and J. Park. Authenticated key exchange secure under the computational Diffie-Hellman assumption. Cryptology ePrint Archive, Report 2008/344, 2008. `http://eprint.iacr.org/`.

[26] R. Marvin. Google admits an Android crypto PRNG flaw led to Bitcoin heist (August 2013). http://sdt.bz/64008 (Accessed 01/10/2013).

[27] A. Menezes, P. van Oorschot, and S. Vanstone. *Handbook of Applied Cryptography*. CRC Press, Boca Raton, FL, USA, 1996.

[28] M. Mueller. Debian OpenSSL predictable PRNG Bruteforce SSH exploit, 2008. http://www.exploit-db.com/exploits/5622/.

[29] N. Perlroth, J. Larson, and S. Scott. NSA able to foil basic safeguards of privacy on web (September 2013). http://www.nytimes.com/2013/09/06/us/nsa-foils-much-internet-encryption.html?pagewanted=1&_r=0 (Accessed 01/10/2013).

[30] T. Ristenpart and S. Yilek. When good randomness goes bad: Virtual machine reset vulnerabilities and hedging deployed cryptography. In *Proceedings of the Network and Distributed System Security Symposium*, NDSS'10, 2010.

[31] B. Schneier. NSA surveillance: A guide to staying secure (September 2013). http://www.theguardian.com/world/2013/sep/05/nsa-how-to-remain-secure-surveillance (Accessed 27/09/2013).

[32] D. Shumow and N. Ferguson. On the Possibility of a Back Door in the NIST SP800-90 Dual Ec Prng. http://rump2007.cr.yp.to/15-shumow.pdf (Accessed 15/10/2013).

[33] B. Ustaoglu. Obtaining a secure and efficient key agreement protocol from (H)MQV and NAXOS. Cryptology ePrint Archive, Report 2007/123, 2007. Version June 22, 2009.

[34] G. Yang, S. Duan, D. S. Wong, C. H. Tan, and H. Wang. Authenticated key exchange under bad randomness. In *Proceedings of the 15th international conference on Financial Cryptography and Data Security*, FC'11, pages 113–126, Berlin, Heidelberg, 2012. Springer-Verlag.

[35] S. Yilek, E. Rescorla, H. Shacham, B. Enright, and S. Savage. When private keys are public: results from the 2008 Debian OpenSSL vulnerability. In *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement conference*, IMC '09, pages 15–27, New York, NY, USA, 2009. ACM.

[36] K. Zetter. How a Crypto 'Backdoor' Pitted the Tech World Against the NSA (September 2013). http://www.wired.com/threatlevel/2013/09/nsa-backdoor/all/ (Accessed 01/10/2013).

A Proof of Proposition 5

Proof. It is straightforward to verify the first condition of Definition 5. We next verify that the second condition of Definition 5 holds. Let E denote a PPT adversary against protocol $\pi := \text{CNX}$. We show that the probability of event Multiple-Match$^{W(\Omega^{-}_{\mathsf{INDP\text{-}DH}})}_{\pi,E}(k)$ is bounded above by a negligible function in the security parameter k, where Multiple-Match$^{W(\Omega^{-}_{\mathsf{INDP\text{-}DH}})}_{\pi,E}(k)$ denotes the event that, in the security experiment, there exist a session s with $s_{status} = \texttt{accepted}$ and at least two distinct sessions s' and s'' that are matching session s. Note that, if both sessions s' and s'' are matching session s, then it must hold that $s''_{actor} = s'_{actor}$ and $s''_{role} = s'_{role}$). In addition, the counter value in two different sessions of the same user are distinct. For some

fixed session s that has accepted, let Ev denote the event that there exist two distinct sessions s' and s'' such that s and s' are matching as well as s and s''. We have:

$$\begin{aligned} P(Ev) &\leq P(\bigcup_{\substack{s',s'' \\ s' \neq s''}} \{H_1(s''_{rand}, \mathsf{sk}_{\hat{P}}, i) = H_1(s'_{rand}, \mathsf{sk}_{\hat{P}}, j)\}) \\ &\leq \sum_{\substack{s',s'' \\ s' \neq s''}} P(\{H_1(s''_{rand}, \mathsf{sk}_{\hat{P}}, i) = H_1(s'_{rand}, \mathsf{sk}_{\hat{P}}, j)\}) \\ &\leq q_s^2 \frac{1}{p}, \end{aligned}$$

where $\hat{P} = s''_{actor} = s'_{actor}$, $i \neq j$ and q_s denotes the number of created sessions (either via the create or the cr-create query) by the adversary. Therefore, $P(\text{Multiple-Match}_{\pi,E}^{W(\Omega_{\mathsf{INDP\text{-}DH}})}(k)) \leq q_s^3 \frac{1}{p}$.

The third condition of Definition 5 is implied by an adaptation of the security proof of NAXOS in the eCK^w model from [18]. Let s^* denote the test session. Consider first the event K^c where the adversary M wins the security experiment against π with non-negligible advantage and does not query H_2 with $(\sigma_1, \sigma_2, \sigma_3, \hat{A}, \hat{B})$, where $\sigma_1 = CDH(Y, A), \sigma_2 = CDH(B, X)$ and $\sigma_3 = CDH(X, Y)$.

Event K^c

If event K^c occurs, then the adversary M must have issued a session-key query to some session s such that $K_s = K_{s^*}$ (where K_s and K_{s^*} denote the session keys computed in sessions s and s^*, respectively) and s does not match s^*. We consider the following four events:

1. A_1 : there exist two distinct sessions s', s'' created via a create query such that $s'_{rand} = s''_{rand}$.
2. A_2 : there exists a session $s \neq s^*$ such that $H_1(s_{rand}, \mathsf{sk}_{s_{actor}}, i) = H_1(s^*_{rand}, \mathsf{sk}_{s^*_{actor}}, j)$.
3. A_3 : there exists a session $s' \neq s^*$ such that $H_2(\text{input}_{s'}) = H_2(\text{input}_{s^*})$ with $\text{input}_{s'} \neq \text{input}_{s^*}$.
4. A_4 : there exists an adversarial query input_M to the oracle H_2 such that $H_2(\text{input}_M) = H_2(\text{input}_{s^*})$ with $\text{input}_M \neq \text{input}_{s^*}$.

In contrast to the NAXOS protocol with respect to model $\Omega_{\mathsf{INDP\text{-}DH}}$, the adversary cannot force two sessions of protocol π of the same user with the same role to compute the same session key via a chosen-randomness replay attack, as the H_1 values in both sessions will be different with overwhelming probability due to different counter values. The latter event is included in event A_2.

Analysis of event K^c

We denote by q_s the number of created sessions (either via the create or the cr-create query) by the adversary and by q_{ro2} the number of queries to the random oracle H_2. We have that

$$\begin{aligned} P(K^c) &\leq P(A_1 \vee A_2 \vee A_3 \vee A_4) \leq P(A_1) + P(A_2) + P(A_3) + P(A_4) \\ &\leq \frac{q_s^2}{2} \frac{1}{2^k} + \frac{q_s}{p} + \frac{q_s + q_{\text{ro2}}}{2^k}, \end{aligned}$$

which is a negligible function of the security parameter k.

In the subsequent events (and their analyses) we assume that no collisions in the queries to the oracle H_1 occur and that none of the events $A_1, ..., A_4$ occurs. As in the proof of [18, Proposition 7], we next consider the following three events:

1. $DL \wedge K$,
2. $T_O \wedge DL^c \wedge K$, and
3. $(T_O)^c \wedge DL^c \wedge K$, where

T_O denotes the event that there exists an origin-session for the test session, DL denotes the event where there exists a user $\hat{C} \in \mathcal{P}$ such that the adversary M, during its execution, queries H_1 with $(*, c, *)$ before issuing a $\mathsf{corrupt}(\hat{C})$ query and K denotes the event that M wins the security experiment against NAXOS by querying H_2 with $(\sigma_1, \sigma_2, \sigma_3, \hat{A}, \hat{B})$, where $\sigma_1 = CDH(Y, A), \sigma_2 = CDH(B, X)$ and $\sigma_3 = CDH(X, Y)$.

Event $DL \wedge K$

Let the input to the GAP-DLog challenge be C. Suppose that event $DL \wedge K$ occurs with non-negligible probability. In this case, the simulator S chooses one user $\hat{C} \in \mathcal{P}$ at random and sets its long-term public key to C. S chooses long-term secret/public key pairs for the remaining honest parties and stores the associated long-term secret keys. Additionally S chooses a random value $m \in_R \{1, 2, ..., q_s\}$. We denote the m'th activated session by adversary M by s^*. Suppose further that $s^*_{actor} = \hat{A}, s^*_{peer} = \hat{B}$ and $s^*_{role} = \mathcal{I}$, w.l.o.g. We now define S's responses to M's queries for the pre-specified peer setting; the post-specified peer case proceeds similarly. Algorithm S maintains tables Q, J, T and L, all of which are initially empty. S also maintains a variable ω initialized with 1 and a table CV maintaining for each user the current counter value. Initially, table CV contains an entry $(\hat{P}, 0)$ for each user $\hat{P} \in \mathcal{P}$.

1. $\mathsf{create}\left(\hat{P}, r, \hat{Q}\right)$ to create session s: S checks whether $\hat{P} \in \mathcal{P}, \hat{Q} \in \mathcal{P}$, and $r \in \{\mathcal{I}, \mathcal{R}\}$. If one of the checks fails, then S returns $\perp$. Else, S initializes the session variables according to the protocol specification, and stores an entry of the form $(s, s_{rand}, \mathsf{sk}_{s_{actor}}, l_s, \kappa) \in (\mathcal{P} \times \mathbb{N}) \times \{0,1\}^k \times (\mathbb{Z}_p \cup \{*\}) \times \mathbb{N} \times \mathbb{Z}_p$ in table Q as follows:
   - S retrieves the counter value c for the user with identifier $\hat{P}$ from table CV, increments c by 1, and updates the counter value for $\hat{P}$ stored in table CV with $c + 1$,
   - S chooses $s_{rand} \in_R \{0,1\}^k$ (i. e. the randomness of session s),
   - S chooses $\kappa \in_R \mathbb{Z}_p$,
   - if $s_{actor} \neq \hat{C}$, then S stores the entry $(s, s_{rand}, \mathsf{sk}_{s_{actor}}, c + 1, \kappa)$ in Q, else S stores the entry $(s, s_{rand}, *, c + 1, \kappa)$ in Q,[2] and
   - if $r = \mathcal{I}$, then S returns the Diffie-Hellman exponential g^κ to M, else S returns $\star$.
2. $\mathsf{cr\text{-}create}\left(\hat{P}, r, str, \hat{Q}\right)$ to create session s: S checks whether $\hat{P} \in \mathcal{P}, \hat{Q} \in \mathcal{P}$, and $r \in \{\mathcal{I}, \mathcal{R}\}$. If one of the checks fails, then S returns $\perp$. Else, S initializes the session variables according to the protocol specification, and stores an entry of the form $(s, s_{rand}, \mathsf{sk}_{s_{actor}}, l_s, \kappa) \in (\mathcal{P} \times \mathbb{N}) \times \{0,1\}^k \times (\mathbb{Z}_p \cup \{*\}) \times \mathbb{N} \times \mathbb{Z}_p$ in table Q as follows:
   - S retrieves the counter value c for the user with identifier $\hat{P}$ from table CV, increments c by 1, and updates the counter value for $\hat{P}$ stored in table CV with $c + 1$,
   - if there is an entry $(r_i, h_i, l_i, \kappa_i)$ in table J such that $r_i = str$, $h_i = \mathsf{sk}_{\hat{P}}$, and $l_i = c + 1$, then S sets $\omega \leftarrow \kappa_i$, else S chooses $\kappa \in_R \mathbb{Z}_p$, and sets $\omega \leftarrow \kappa$.[3]
   - if $s_{actor} \neq \hat{C}$, then S stores the entry $(s, s_{rand}, \mathsf{sk}_{s_{actor}}, c + 1, x_5)$ in Q, else S stores the entry $(s, s_{rand}, *, c + 1, x_5)$ in Q, where x_5 denotes the value of variable ω,
   - if $r = \mathcal{I}$, then S returns the Diffie-Hellman exponential g^κ to M, else S returns $\star$.
3. S stores entries of the form $(r, h, l, \kappa) \in \{0,1\}^k \times \mathbb{Z}_p \times \mathbb{N} \times \mathbb{Z}_p$ in table J. When M makes a query of the form (r, h, l) to the random oracle for H_1, answer it as follows:
   - If $C = g^h$, then S aborts M and is successful by outputting $\mathrm{DLog}_g(C) = h$.
   - Else if $(r, h, l, \kappa) \in J$ for some $\kappa \in \mathbb{Z}_p$, then S returns κ to M.
   - Else if there exists an entry $(s, s_{rand}, \mathsf{sk}_{s_{actor}}, l_s, \kappa)$ in Q, for some $s \in \mathcal{P} \times \mathbb{N}, s_{rand} \in \{0,1\}^k, \mathsf{sk}_{s_{actor}} \in \mathbb{Z}_p$, $l_s \in \mathbb{N}$ and $\kappa \in \mathbb{Z}_p$, such that $s_{rand} = r$, $\mathsf{sk}_{s_{actor}} = h$ and $l_s = l$, then S returns κ to M and stores the entry (r, h, l, κ) in table J.
   - Else, S chooses $\kappa \in_R \mathbb{Z}_p$, returns it to M and stores the entry (r, h, l, κ) in table J.
4. $\mathsf{send}(\hat{P}, i, V)$ to send message V to session $s = (\hat{P}, i)$: If $s_{status} \neq \mathtt{active}$, then S returns $\perp$.

[2] We do not need to keep consistency with H_1 queries via lookup in table J since the probability that the adversary guesses the randomness of a session created via a query create is negligible.

[3] Here we need to keep consistency with H_1 queries via lookup in table J to be able to consistently answer all possible combinations of queries. Consider, e. g., the following scenario. The adversary first issues a query $(x, \mathsf{sk}_{\hat{P}}, i)$ to H_1 and then issues the query $\mathsf{cr\text{-}create}(\hat{P}, r, x, \hat{Q})$, which increments the current counter value $i - 1$ by 1 so that the counter value used in session $s = (\hat{P}, i)$ is i. So, in contrast to the NAXOS proof with respect to model eCK^w, we need to additionally keep consistency between $\mathsf{cr\text{-}create}$ queries and queries to the random oracle for H_1.

Else if $s_{role} = \mathcal{I}$, then S does the following. If $V \notin G$, then the status of session s is set to `rejected`. Else, the status of session s is set to `accepted`, the variable $recv$ is updated to $s_{recv} \leftarrow (s_{recv}, V)$ and

- If there exists an entry $(s_{peer}, s_{actor}, \mathcal{R}, s_{recv}, s_{sent}, \lambda)$ in table T, then S stores the entry $(s_{actor}, s_{peer}, \mathcal{I}, s_{sent}, s_{recv}, \lambda)$ in table T.
- Else if there exists an entry $(\sigma_1, \sigma_2, \sigma_3, s_{actor}, s_{peer}, \lambda)$ in table L, for some $\lambda \in \{0,1\}^k$, such that $\mathrm{DDH}(s_{recv}, s_{sent}, \sigma_3) = 1$, $\mathrm{DDH}(s_{sent}, \mathsf{pk}_{s_{peer}}, \sigma_2) = 1$ and $\mathrm{DDH}(s_{recv}, \mathsf{pk}_{s_{actor}}, \sigma_1) = 1$, then S stores $(s_{actor}, s_{peer}, \mathcal{I}, s_{sent}, s_{recv}, \lambda)$ in table T.
- Else, S chooses $\mu \in_R \{0,1\}^k$ and stores the entry $(s_{actor}, s_{peer}, \mathcal{I}, s_{sent}, s_{recv}, \mu)$ in T.

Else if $s_{role} = \mathcal{R}$, then S does the following. If $V \notin G$, then the status of session s is set to `rejected`. Else, S sets the status of session s to `accepted`, and the variable $recv$ to (s_{recv}, V). S returns g^{κ} to M, where κ denotes the last element of the entry $(s, r, \mathsf{sk}_{s_{actor}}, l, \kappa)$ in table Q, and proceeds in a similar way as in the previous case.

5. When M makes a query of the form $\left(\sigma_1, \sigma_2, \sigma_3, \hat{P}_i, \hat{P}_j\right)$ to the random oracle for H_2, answer it as follows:
   - If $\left(\sigma_1, \sigma_2, \sigma_3, \hat{P}_i, \hat{P}_j, \lambda\right) \in L$ for some $\lambda \in \{0,1\}^k$, then S returns λ to M.
   - Else if there exist entries $\left(\hat{P}_i, \hat{P}_j, \mathcal{I}, U, V, \lambda\right)$ or $\left(\hat{P}_j, \hat{P}_i, \mathcal{R}, V, U, \lambda\right)$ in table T, for some $\lambda \in \{0,1\}^k$ and $U, V \in G$, such that $\mathrm{DDH}(V, U, \sigma_3) = 1$, $\mathrm{DDH}(V, \mathsf{pk}_{\hat{P}_i}, \sigma_1) = 1$ and $\mathrm{DDH}(U, \mathsf{pk}_{\hat{P}_j}, \sigma_2) = 1$, then S returns λ to M and stores the entry $\left(\sigma_1, \sigma_2, \sigma_3, \hat{P}_i, \hat{P}_j, \lambda\right)$ in table L.
   - Else, S chooses $\mu \in_R \{0,1\}^k$, returns it to M and stores the entry $\left(\sigma_1, \sigma_2, \sigma_3, \hat{P}_i, \hat{P}_j, \mu\right)$ in L.
6. $\mathsf{randomness}(s)$: If $s_{status} = \perp$, then S returns $\perp$. Otherwise, S returns s_{rand} (via lookup in table Q).
7. $\mathsf{session\text{-}key}(s)$: If $s_{status} \neq \mathtt{accepted}$, then S returns $\perp$. Otherwise, S answers this query by lookup in table T.
8. $\mathsf{test\text{-}session}(s)$: If $s \neq s^*$, then S aborts; otherwise S answers the query in the appropriate way.
9. $\mathsf{corrupt}(\hat{P})$: If $\hat{P} \notin \mathcal{P}$, then S returns $\perp$. Else if $\hat{P} = \hat{C}$, then S aborts. Else S returns $\mathsf{sk}_{\hat{P}}$.
10. M outputs a guess: S aborts.

Analysis of event $DL \wedge K$
Similar to the analysis of the related event $DL \wedge K$ in the proof of [18, Proposition 7].

Event $T_O \wedge DL^c \wedge K$
Let s^* and s' denote the test session and the origin-session for the test session, respectively. We split event $Evt := T_O \wedge DL^c \wedge K$ into the following events $B_1, ..., B_3$ so that $Evt = B_1 \vee B_2 \vee B_3$:

1. B_1 : Evt occurs and $s^*_{peer} = s'_{actor}$.
2. B_2 : Evt occurs and $s^*_{peer} \neq s'_{actor}$ and M does issue neither a $\mathsf{randomness}(s')$ query nor a $\mathsf{cr\text{-}create}(s', \times)$ query to the origin-session s' of s^*, but may issue a $\mathsf{corrupt}(s^*_{peer})$ query.
3. B_3 : Evt occurs and $s^*_{peer} \neq s'_{actor}$ and M does not issue a $\mathsf{corrupt}(s^*_{peer})$ query, but may issue either a $\mathsf{randomness}(s')$ query or a $\mathsf{cr\text{-}create}(s', \times)$ query to the origin-session s' of s^*.

Event B_1
Let the input to the GDH challenge be (X_0, Y_0). Suppose that event B_1 occurs with non-negligible probability. In this case S chooses long-term secret/public key pairs for all the honest parties and stores the associated long-term secret keys. Additionally S chooses two random values $m, n \in_R \{1, 2, ..., q_s\}$. The m'th activated session by adversary M will be called s^* and the n'th activated session will be called s'. Suppose further that $s^*_{actor} = \hat{A}, s^*_{peer} = \hat{B}$ and $s^*_{role} = \mathcal{I}$,

w.l.o.g.. The simulation of M's environment proceeds as follows:

1. $\mathsf{create}(\hat{A}, \mathcal{I}, \hat{B})$ or $\mathsf{cr\text{-}create}(\hat{A}, \mathcal{I}, str, \hat{B})$ to create session s^*: If create is issued, then S chooses $s^*_{rand} \in_R \{0,1\}^k$. Else, S sets $s^*_{rand} \leftarrow str$. Then, S (a) returns the message X_0, where (X_0, Y_0) is the GDH challenge, (b) increments by 1 the counter value c for the user with identifier $\hat{A}$ (stored in table CV), and (c) stores the updated counter value $c+1$ for $\hat{A}$ in table CV.[4]
2. $\mathsf{create}(\hat{B}, r, \hat{Q})$ or $\mathsf{cr\text{-}create}(\hat{B}, r, str, \hat{Q})$ to create session s': If create is issued, then S chooses $s'_{rand} \in_R \{0,1\}^k$. Else, S sets $s'_{rand} \leftarrow str$. S then increments by 1 the counter value c for the user with identifier $\hat{B}$ (stored in table CV), and stores the updated counter value $c+1$ for $\hat{B}$ in table CV.If $r = \mathcal{I}$, then S returns message Y_0 to M, where (X_0, Y_0) is the GDH challenge. Else, $\star$ is returned.
3. $\mathsf{send}(\hat{B}, i, Z)$ with $(\hat{B}, i) = s'$: If $s'_{status} \neq \mathtt{active}$, then S returns $\perp$. Else if $s'_{role} = \mathcal{R}$ and $Z \in G$, then S returns message Y_0 to M, where (X_0, Y_0) is the GDH challenge, sets the status of session s' to $\mathtt{accepted}$, and proceeds as in the previous simulation for completing the session. Else, S proceeds as in the previous simulation.
4. $\mathsf{send}(\hat{A}, i, Y_0)$ with $(\hat{A}, i) = s^*$: S proceeds as in the previous simulation for completing the session.
5. Other $\mathsf{create}, \mathsf{cr\text{-}create}$ and send queries are answered as in the previous simulation.
6. $\mathsf{randomness}(s)$: If $s_{status} = \perp$, then S returns $\perp$. Else, S returns s_{rand}.
7. $\mathsf{session\text{-}key}(s)$: If $s_{status} \neq \mathtt{accepted}$, then S returns $\perp$. Otherwise, S answers this query by lookup in table T.
8. $\mathsf{test\text{-}session}(s)$: If $s \neq s^*$ or if s' is not the origin-session for session s^*, then S aborts; otherwise S answers the query in the appropriate way.
9. $H_1(r, h, *)$: If $h = a$ and $r = s^*_{rand}$ or if $h = b$ and $r = s'_{rand}$, then S aborts. Otherwise S simulates a random oracle as in the previous simulation.
10. $\mathsf{corrupt}(\hat{P})$: If $\hat{P} \notin \mathcal{P}$, then S returns $\perp$. Else, S returns $\mathsf{sk}_{\hat{P}}$.
11. When M makes a query of the form $\left(\sigma_1, \sigma_2, \sigma_3, \hat{P}_i, \hat{P}_j\right)$ to the random oracle for H_2, answer it as follows:
    - If $\left\{\hat{P}_i, \hat{P}_j\right\} = \left\{\hat{A}, \hat{B}\right\}$, $\sigma_1 = Y_0^a$, $\sigma_2 = X_0^b$ and $\mathrm{DDH}(X_0, Y_0, \sigma_3) = 1$, then S aborts M and is successful by outputting $CDH(X_0, Y_0) = \sigma_3$.
    - Else if $\left(\sigma_1, \sigma_2, \sigma_3, \hat{P}_i, \hat{P}_j, \lambda\right) \in L$ for some $\lambda \in \{0,1\}^k$, then S returns λ to M.
    - Else if there exist entries $\left(\hat{P}_i, \hat{P}_j, \mathcal{I}, U, V, \lambda\right)$ or $\left(\hat{P}_j, \hat{P}_i, \mathcal{R}, V, U, \lambda\right)$ in table T, for some $\lambda \in \{0,1\}^k$ and $U, V \in G$, such that $\mathrm{DDH}(V, U, \sigma_3) = 1$, $\mathrm{DDH}(V, \mathsf{pk}_{\hat{P}_i}, \sigma_1) = 1$ and $\mathrm{DDH}(U, \mathsf{pk}_{\hat{P}_j}, \sigma_2) = 1$, then S returns λ to M and stores the entry $\left(\sigma_1, \sigma_2, \sigma_3, \hat{P}_i, \hat{P}_j, \lambda\right)$ in table L.
    - Else, S chooses $\mu \in_R \{0,1\}^k$, returns it to M and stores the entry $\left(\sigma_1, \sigma_2, \sigma_3, \hat{P}_i, \hat{P}_j, \mu\right)$ in L.
12. M outputs a guess: S aborts.

Analysis of event B_1
Similar to the analysis of the related event B_1 in the proof of [18, Proposition 7].

Event B_2
Let the input to the GDH challenge be (X_0, Y_0). Suppose that event B_2 occurs with non-negligible probability. The simulation of S proceeds in the same way as for event B_1 with the following changes:

- $\mathsf{create}(\hat{B}, r, \hat{Q})$ or $\mathsf{cr\text{-}create}(\hat{B}, r, str, \hat{Q})$ to create session s': If $\mathsf{cr\text{-}create}$ is issued, then S aborts. Else, S proceeds as described before.

[4] Note that s^*_{rand} is not used in the calculation.

- $\mathsf{randomness}(s)$: If $s_{status} = \perp$, then S returns $\perp$. Else if $s = s'$, then S aborts. Else, S returns s_{rand}.
- $H_1(r, h, *)$: If $h = a$ and $r = s^*_{rand}$, then S aborts. Otherwise S simulates a random oracle as in the previous simulation.

Analysis of event B_2
Similar to the analysis of the related event B_2 in the proof of [18, Proposition 7].

Event B_3
Let the input to the GDH challenge be (X_0, B). Suppose that event B_3 occurs with non-negligible probability. In this case, S chooses one user $\hat{B} \in \mathcal{P}$ at random from the set $\mathcal{P}$ and sets its long-term public key to B. S chooses long-term secret/public key pairs for the remaining parties in $\mathcal{P}$ and stores the associated long-term secret keys. Additionally S chooses two random values $m, n \in_R \{1, 2, ..., q_s\}$. We denote the m'th activated session by adversary M by s^* and the n'th activated session by s'. Suppose further that $s^*_{actor} = \hat{A}, s^*_{peer} = \hat{B}$ and $s^*_{role} = \mathcal{I}$, w.l.o.g.. Algorithm S maintains tables Q, J, T and L, all of which are initially empty. S also maintains a variable ω initialized with 1 and a table CV maintaining for each user the current counter value. Initially, table CV contains an entry $(\hat{P}, 0)$ for each user $\hat{P} \in \mathcal{P}$. The simulation of M's environment proceeds as follows:

1. $\mathsf{create}(\hat{A}, \mathcal{I}, \hat{B})$ or $\mathsf{cr\text{-}create}(\hat{A}, \mathcal{I}, str, \hat{B})$ to create session s^*: If create is issued, then S chooses $s^*_{rand} \in_R \{0,1\}^k$. Else, S sets $s^*_{rand} \leftarrow str$. Then, S (a) returns the message X_0, (b) increments by 1 the counter value c for the user with identifier $\hat{A}$ (stored in table CV), and (c) stores the updated counter value $c + 1$ for $\hat{A}$ in table CV.
2. $\mathsf{create}\left(\hat{P}, r, \hat{Q}\right)$ to create session s: S checks whether $\hat{P} \in \mathcal{P}, \hat{Q} \in \mathcal{P}$, and $r \in \{\mathcal{I}, \mathcal{R}\}$. If one of the checks fails, then S returns $\perp$. Else, S initializes the session variables according to the protocol specification, and stores an entry of the form $(s, s_{rand}, \mathsf{sk}_{s_{actor}}, l_s, \kappa) \in (\mathcal{P} \times \mathbb{N}) \times \{0,1\}^k \times (\mathbb{Z}_p \cup \{*\}) \times \mathbb{N} \times \mathbb{Z}_p$ in table Q as follows:
   - S retrieves the counter value c for the user with identifier $\hat{P}$ from table CV, increments c by 1, and updates the counter value for $\hat{P}$ stored in table CV with $c + 1$,
   - S chooses $s_{rand} \in_R \{0,1\}^k$ (i. e. the randomness of session s),
   - S chooses $\kappa \in_R \mathbb{Z}_p$,
   - if $s_{actor} \neq \hat{B}$, then S stores the entry $(s, s_{rand}, \mathsf{sk}_{s_{actor}}, c + 1, \kappa)$ in Q, else S stores the entry $(s, s_{rand}, *, c + 1, \kappa)$ in Q, and
   - if $r = \mathcal{I}$, then S returns the Diffie-Hellman exponential g^κ to M, else S returns $\star$.
3. $\mathsf{cr\text{-}create}\left(\hat{P}, r, str, \hat{Q}\right)$ to create session s: S checks whether $\hat{P} \in \mathcal{P}, \hat{Q} \in \mathcal{P}$, and $r \in \{\mathcal{I}, \mathcal{R}\}$. If one of the checks fails, then S returns $\perp$. Else, S initializes the session variables according to the protocol specification, and stores an entry of the form $(s, s_{rand}, \mathsf{sk}_{s_{actor}}, l_s, \kappa) \in (\mathcal{P} \times \mathbb{N}) \times \{0,1\}^k \times (\mathbb{Z}_p \cup \{*\}) \times \mathbb{N} \times \mathbb{Z}_p$ in table Q as follows:
   - S retrieves the counter value c for the user with identifier $\hat{P}$ from table CV, increments c by 1, and updates the counter value for $\hat{P}$ stored in table CV with $c + 1$,
   - if there is an entry $(r_i, h_i, l_i, \kappa_i)$ in table J such that $r_i = str$, $h_i = \mathsf{sk}_{\hat{P}}$, and $l_i = c + 1$, then S sets $\omega \leftarrow \kappa_i$, else S chooses $\kappa \in_R \mathbb{Z}_p$, and sets $\omega \leftarrow \kappa$.
   - if $s_{actor} \neq \hat{B}$, then S stores the entry $(s, s_{rand}, \mathsf{sk}_{s_{actor}}, c + 1, x_5)$ in Q, else S stores the entry $(s, s_{rand}, *, c + 1, x_5)$ in Q, where x_5 denotes the value of variable ω,
   - if $r = \mathcal{I}$, then S returns the Diffie-Hellman exponential g^κ to M, else S returns $\star$.
4. S stores entries of the form $(r, h, l, \kappa) \in \{0,1\}^k \times \mathbb{Z}_p \times \mathbb{N} \times \mathbb{Z}_p$ in table J. When M makes a query of the form (r, h, l) to the random oracle for H_1, answer it as follows:
   - If $r = s^*_{rand}$ and $h = a$, then S aborts,
   - Else if $(r, h, l, \kappa) \in J$ for some $\kappa \in \mathbb{Z}_p$, then S returns κ to M.
   - Else if there exists an entry $(s, s_{rand}, \mathsf{sk}_{s_{actor}}, l_s, \kappa)$ in Q, for some $s \in \mathcal{P} \times \mathbb{N}, s_{rand} \in \{0,1\}^k, \mathsf{sk}_{s_{actor}} \in \mathbb{Z}_p$, $l_s \in \mathbb{N}$ and $\kappa \in \mathbb{Z}_p$, such that $s_{rand} = r$, $\mathsf{sk}_{s_{actor}} = h$ and $l_s = l$,

then S returns κ to M and stores the entry (r, h, l, κ) in table J.
- Else, S chooses $\kappa \in_R \mathbb{Z}_p$, returns it to M and stores the entry (r, h, l, κ) in table J.

5. $\mathsf{send}(\hat{P}, i, V)$ to send message V to session $s = (\hat{P}, i)$: If $s_{status} \neq \texttt{active}$, then S returns $\perp$. Else if $s_{role} = \mathcal{I}$, then S does the following. If $V \notin G$, then the status of session s is set to `rejected`. Else, the status of session s is set to `accepted`, the variable $recv$ is updated to $s_{recv} \leftarrow (s_{recv}, V)$ and
   - If there exists an entry $(s_{peer}, s_{actor}, \mathcal{R}, s_{recv}, s_{sent}, \lambda)$ in table T, then S stores the entry $(s_{actor}, s_{peer}, \mathcal{I}, s_{sent}, s_{recv}, \lambda)$ in table T.
   - Else if there exists an entry $(\sigma_1, \sigma_2, \sigma_3, s_{actor}, s_{peer}, \lambda)$ in table L, for some $\lambda \in \{0,1\}^k$, such that $\mathrm{DDH}(s_{recv}, s_{sent}, \sigma_3) = 1$, $\mathrm{DDH}(s_{sent}, \mathsf{pk}_{s_{peer}}, \sigma_2) = 1$ and $\mathrm{DDH}(s_{recv}, \mathsf{pk}_{s_{actor}}, \sigma_1) = 1$, then S stores $(s_{actor}, s_{peer}, \mathcal{I}, s_{sent}, s_{recv}, \lambda)$ in table T.
   - Else, S chooses $\mu \in_R \{0,1\}^k$ and stores the entry $(s_{actor}, s_{peer}, \mathcal{I}, s_{sent}, s_{recv}, \mu)$ in T.

   Else if $s_{role} = \mathcal{R}$, then S does the following. If $V \notin G$, then the status of session s is set to `rejected`. Else, S sets the status of session s to `accepted`, and the variable $recv$ to (s_{recv}, V). S returns g^κ to M, where κ denotes the last element of the entry $(s, r, \mathsf{sk}_{s_{actor}}, l, \kappa)$ in table Q, and proceeds in a similar way as in the previous case.
6. When M makes a query of the form $\left(\sigma_1, \sigma_2, \sigma_3, \hat{P}_i, \hat{P}_j\right)$ to the random oracle for H_2, answer it as follows:
   - If $s'_{status} \neq \perp$, $\left\{\hat{P}_i, \hat{P}_j\right\} = \left\{\hat{A}, \hat{B}\right\}$, $\sigma_1 = A^\kappa$, $\mathrm{DDH}(X_0, B, \sigma_2) = 1$, and $\sigma_3 = X_0^\kappa$, where κ denotes the last element of the entry $(s', s'_{rand}, \mathsf{sk}_{s'_{actor}}, l, \kappa)$ in table Q[5], then S aborts M and is successful by outputting $CDH(X_0, B) = \sigma_2$.
   - Else if $\left(\sigma_1, \sigma_2, \sigma_3, \hat{P}_i, \hat{P}_j, \lambda\right) \in L$ for some $\lambda \in \{0,1\}^k$, then S returns λ to M.
   - Else if there exist entries $\left(\hat{P}_i, \hat{P}_j, \mathcal{I}, U, V, \lambda\right)$ or $\left(\hat{P}_j, \hat{P}_i, \mathcal{R}, V, U, \lambda\right)$ in table T, for some $\lambda \in \{0,1\}^k$ and $U, V \in G$, such that $\mathrm{DDH}(V, U, \sigma_3) = 1$, $\mathrm{DDH}(V, \mathsf{pk}_{\hat{P}_i}, \sigma_1) = 1$ and $\mathrm{DDH}(U, \mathsf{pk}_{\hat{P}_j}, \sigma_2) = 1$, then S returns λ to M and stores the entry $\left(\sigma_1, \sigma_2, \sigma_3, \hat{P}_i, \hat{P}_j, \lambda\right)$ in table L.
   - Else, S chooses $\mu \in_R \{0,1\}^k$, returns it to M and stores the entry $\left(\sigma_1, \sigma_2, \sigma_3, \hat{P}_i, \hat{P}_j, \mu\right)$ in L.
7. $\mathsf{randomness}(s)$: If $s_{status} = \perp$, then S returns $\perp$. Else, S returns s_{rand}.
8. $\mathsf{session\text{-}key}(s)$: If $s_{status} \neq \texttt{accepted}$, then S returns $\perp$. Otherwise, S answers this query by lookup in table T.
9. $\mathsf{test\text{-}session}(s)$: If $s \neq s^*$ or if s' is not the origin-session for session s^*, then S aborts; otherwise S answers the query in the appropriate way.
10. $\mathsf{corrupt}(\hat{P})$: If $\hat{P} \notin \mathcal{P}$, then S returns $\perp$. Else if $\hat{P} = \hat{B}$, then S aborts. Else, S returns $\mathsf{sk}_{\hat{P}}$.
11. M outputs a guess: S aborts.

Analysis of event B_3
Similar to the analysis of the related event B_3 in the proof of [18, Proposition 7].

Event $(T_O)^c \wedge DL^c \wedge K$
The simulation and analysis are very similar to the simulation and analysis related to event B_3. □

[5]This entry exists in table Q since the status of the session is different to $\perp$.

B Proof of Proposition 7

Proof. It is straightforward to verify the first condition of Definition 5. We next verify that the second condition of Definition 5 holds. Let E denote a PPT adversary against protocol $\pi := \text{NXPR}$. We show that the probability of event Multiple-Match$_{\pi,E}^{W(\Omega_{\mathsf{INDP\text{-}DH}\cap\mathsf{ISM}})}(k)$ is bounded above by a negligible function in the security parameter k, where Multiple-Match$_{\pi,E}^{W(\Omega_{\mathsf{INDP\text{-}DH}\cap\mathsf{ISM}})}(k)$ denotes the event that, in the security experiment, there exist a session s with $s_{status} = \mathtt{accepted}$ and at least two distinct sessions s' and s'' that are matching session s. Note that, if both sessions s' and s'' are matching session s, then it must hold that $s''_{actor} = s'_{actor}$ and $s''_{role} = s'_{role}$. In addition, it is easy to see that the value of the variable *data* in two different sessions of the same user are distinct (since of different length). For some fixed session s that has accepted, let Ev denote the event that there exist two distinct sessions s' and s'' such that s and s' are matching as well as s and s''. We have:

$$\begin{aligned} P(Ev) &\leq P(\bigcup_{\substack{s',s'' \\ s'\neq s''}} \{H_1(s''_{rand}, s''_{data}, \mathsf{sk}_{\hat{P}}) = H_1(s'_{rand}, s'_{data}, \mathsf{sk}_{\hat{P}})\}) \\ &\leq \sum_{\substack{s',s'' \\ s'\neq s''}} P(\{H_1(s''_{rand}, s''_{data}, \mathsf{sk}_{\hat{P}}) = H_1(s'_{rand}, s'_{data}, \mathsf{sk}_{\hat{P}})\}) \\ &\leq \frac{q_s^2}{p}, \end{aligned}$$

where $\hat{P} = s''_{actor} = s'_{actor}$ and q_s denotes the number of created sessions (either via the create or the cr-create query) by the adversary.

In the above computation, we distinguished between the following two events:

1. $D_1 := \{s''_{rand} \neq s'_{rand} \wedge s''_{data} \neq s'_{data}\}$; the probability that the two hash values are identical given D_1 is the probability of a collision in the hash function, and
2. $D_2 := \{s''_{rand} = s'_{rand} \wedge s''_{data} \neq s'_{data}\}$; the probability that the two hash values are identical given D_2 is the probability of a collision in the hash function.

The events $D_3 := \{s''_{rand} = s'_{rand} \wedge s''_{data} = s'_{data}\}$ and $D_4 := \{s''_{rand} \neq s'_{rand} \wedge s''_{data} = s'_{data}\}$ both occur with probability zero.

Even though the value of the variable *rand* can be the same for two different session of the same user due to the queries cr-create and randomness, the value of the variable *data* of two different sessions s' and s'' of the same user is always different since the bit strings s'_{data} and s''_{data} differ in length. Given a created session s, the length of the bit string s_{data} depends on the number of sessions of user s_{actor} that have already been created either via create or cr-create.

Finally, $P(\text{Multiple-Match}_{\pi,E}^{W(\Omega_{\mathsf{INDP\text{-}DH}\cap\mathsf{ISM}})}(k)) \leq q_s^3 \frac{1}{p}$.

The third condition of Definition 5 is implied by an adaptation of the security proof of protocol CNX in the $\Omega^{-}_{\mathsf{INDP\text{-}DH}}$ model (see Appendix A). Let s^* denote the test session. Consider first the event K^c where the adversary M wins the security experiment against π with non-negligible advantage and does not query H_2 with $(\sigma_1, \sigma_2, \sigma_3, \hat{A}, \hat{B})$, where $\sigma_1 = CDH(Y, A), \sigma_2 = CDH(B, X)$ and $\sigma_3 = CDH(X, Y)$.

Event K^c

If event K^c occurs, then the adversary M must have issued a session-key query to some session s such that $K_s = K_{s^*}$ (where K_s and K_{s^*} denote the session keys computed in sessions s and s^*, respectively) and s does not match s^*. We consider the following four events:

1. A_1 : there exist two distinct sessions s', s'' created via a create query such that $s'_{rand} = s''_{rand}$.[6]
2. A_2 : there exists a session $s \neq s^*$ such that $H_1(s_{rand}, s_{data}) = H_1(s^*_{rand}, s^*_{data})$.
3. A_3 : there exists a session $s' \neq s^*$ such that $H_2(\text{input}_{s'}) = H_2(\text{input}_{s^*})$ with $\text{input}_{s'} \neq \text{input}_{s^*}$.
4. A_4 : there exists an adversarial query input_M to the oracle H_2 such that $H_2(\text{input}_M) = H_2(\text{input}_{s^*})$ with $\text{input}_M \neq \text{input}_{s^*}$.

[6] Under event A_1 the query randomness (e. g., for two sessions of different users) together with other queries might enable the adversary to learn all the information necessary to compute the session key of the target session without violating the freshness condition.

Analysis of event K^c

We denote by q_s the number of created sessions (either via the query create or the query cr-create) by the adversary and by q_{ro2} the number of queries to the random oracle H_2. We have that

$$\begin{aligned} P(K^c) &\leq P(A_1 \vee A_2 \vee A_3 \vee A_4) \leq P(A_1) + P(A_2) + P(A_3) + P(A_4) \\ &\leq \frac{q_s^2}{2}\frac{1}{2^k} + \frac{q_s}{p} + \frac{q_s + q_{\text{ro2}}}{2^k}, \end{aligned}$$

which is a negligible function of the security parameter k.

In contrast to the NAXOS protocol analyzed with respect to model $\Omega_{\mathsf{INDP\text{-}DH}}$, the adversary cannot force two sessions of protocol π of the same user with the same role to compute the same session key via a chosen-randomness replay attack since the H_1 values in both sessions will be different with overwhelming probability. The latter event is included in event A_2.

In the subsequent events (and their analyses) we assume that no collisions in the queries to the oracle H_1 occur and that none of the events $A_1, ..., A_4$ occurs. As in the proof of [18, Proposition 7], we next consider the following three events:

1. $DL \wedge K$,
2. $T_O \wedge DL^c \wedge K$, and
3. $(T_O)^c \wedge DL^c \wedge K$, where

T_O denotes the event that there exists an origin-session for the test session, DL denotes the event where there exists a user $\hat{C} \in \mathcal{P}$ such that the adversary M, during its execution, queries H_1 with $(*, c)$ before issuing a $\mathsf{corrupt}(\hat{C})$ query and K denotes the event that M wins the security experiment against NXPR by querying H_2 with $(\sigma_1, \sigma_2, \sigma_3, \hat{A}, \hat{B})$, where $\sigma_1 = CDH(Y, A), \sigma_2 = CDH(B, X)$ and $\sigma_3 = CDH(X, Y)$.

Event $DL \wedge K$

Let the input to the GAP-DLog challenge be C. Suppose that event $DL \wedge K$ occurs with non-negligible probability. In this case, the simulator S chooses one user $\hat{C} \in \mathcal{P}$ at random and sets its long-term public key to C. S chooses long-term secret/public key pairs for the remaining honest parties and stores the associated long-term secret keys. Additionally S chooses a random value $m \in_R \{1, 2, ..., q_s\}$. We denote the m'th activated session by adversary M by s^*. Suppose further that $s^*_{actor} = \hat{A}, s^*_{peer} = \hat{B}$ and $s^*_{role} = \mathcal{I}$, w.l.o.g. We now define S's responses to M's queries for the pre-specified peer setting; the post-specified peer case proceeds similarly. Algorithm S maintains tables Q, J, T and L, all of which are initially empty. S also maintains a variable ω initialized with 1.

1. $\mathsf{create}\left(\hat{P}, r, \hat{Q}\right)$ to create session s: S checks whether $\hat{P} \in \mathcal{P}$, $\hat{Q} \in \mathcal{P}$, and $r \in \{\mathcal{I}, \mathcal{R}\}$. If one of the checks fails, then S returns $\perp$. Else, S initializes the session variables according to the protocol specification, and stores an entry of the form $(s, s_{rand}, l_s, \mathsf{sk}_{s_{actor}}, \kappa) \in (\mathcal{P} \times \mathbb{N}) \times \{0,1\}^k \times \{0,1\}^* \times (\mathbb{Z}_p \cup \{*\}) \times \mathbb{Z}_p$ in table Q as follows:
   - S chooses $s_{rand} \in_R \{0,1\}^k$ (i. e. the randomness of session s),
   - S chooses $\kappa \in_R \mathbb{Z}_p$,
   - if there is no entry $(s, s_{rand}, l_s, \mathsf{sk}_{s_{actor}}, \kappa)$ in table Q such that $s_{actor} = \hat{P}$, then S sets the value of l_s to s_{rand}, else S sets the value of l_s to $(s_{rand}, l_{s'})$, where s' is the previous session with $s'_{actor} = s_{actor}$ for which an entry in table Q has been made.[7]
   - if $s_{actor} \neq \hat{C}$, then S stores the entry $(s, s_{rand}, s_{data}, \mathsf{sk}_{s_{actor}}, \kappa)$ in Q, else S stores the entry $(s, s_{rand}, s_{data}, *, \kappa)$ in Q, and
   - if $r = \mathcal{I}$, then S returns the Diffie-Hellman exponential g^κ to M, else S returns $\star$.
2. $\mathsf{cr\text{-}create}\left(\hat{P}, r, str, \hat{Q}\right)$ to create session s: S checks whether $\hat{P} \in \mathcal{P}$, $\hat{Q} \in \mathcal{P}$, and $r \in \{\mathcal{I}, \mathcal{R}\}$. If one of the checks fails, then S returns $\perp$. Else, S initializes the session variables according

[7] The value of $l_{s'}$ is the concatenation of the randomness of the current and the previous sessions of the same user.

to the protocol specification, and stores an entry of the form $(s, s_{rand}, l_s, \mathsf{sk}_{s_{actor}}, \kappa) \in (\mathcal{P} \times \mathbb{N}) \times \{0,1\}^k \times \{0,1\}^* \times (\mathbb{Z}_p \cup \{*\}) \times \mathbb{Z}_p$ in table Q as follows:

- if there is an entry (r_i, h_i, κ_i) in table J such that $r_i = (str, l_{s'})$, and $h_i = \mathsf{sk}_{\hat{P}}$, where s' is the previous session with $s'_{actor} = s_{actor}$ for which an entry in table Q has been made, then S sets $\omega \leftarrow \kappa_i$, else S chooses $\kappa \in_R \mathbb{Z}_p$ and sets $\omega \leftarrow \kappa$.
- if $s_{actor} \neq \hat{C}$, then S stores the entry $(s, s_{rand}, r_i, \mathsf{sk}_{s_{actor}}, \omega)$ in Q, else S stores the entry $(s, s_{rand}, l_s, *, \omega)$ in Q with $l_s = (str, l_{s'})$,
- if $r = \mathcal{I}$, then S returns the Diffie-Hellman exponential g^κ to M, else S returns $\star$.

3. S stores entries of the form $(r, h, \kappa) \in \{0,1\}^* \times \mathbb{Z}_p \times \mathbb{Z}_p$ in table J. When M makes a query of the form (r, h) to the random oracle for H_1, answer it as follows:
   - If $C = g^h$, then S aborts M and is successful by outputting $\mathrm{DLog}_g(C) = h$.
   - Else if $(r, h, \kappa) \in J$ for some $\kappa \in \mathbb{Z}_p$, then S returns κ to M.
   - Else if there exists an entry $(s, s_{rand}, l_s, \mathsf{sk}_{s_{actor}}, \kappa)$ in table Q with $l_s = r$ and $\mathsf{sk}_{s_{actor}} = h$, then S returns κ to M and stores the entry (r, h, κ) in table J.
   - Else, S chooses $\kappa \in_R \mathbb{Z}_p$, returns it to M and stores the entry (r, h, κ) in table J.
4. $\mathsf{send}(\hat{P}, i, V)$ to send message V to session $s = (\hat{P}, i)$: If $s_{status} \neq \texttt{active}$, then S returns $\perp$. Else if $s_{role} = \mathcal{I}$, then S does the following. If $V \notin G$, then the status of session s is set to `rejected`. Else, the status of session s is set to `accepted`, and
   - If there exists an entry $(s_{peer}, s_{actor}, \mathcal{R}, s_{recv}, s_{sent}, \lambda)$ in table T, then S stores the entry $(s_{actor}, s_{peer}, \mathcal{I}, s_{sent}, s_{recv}, \lambda)$ in table T.
   - Else if there exists an entry $(\sigma_1, \sigma_2, \sigma_3, s_{actor}, s_{peer}, \lambda)$ in table L, for some $\lambda \in \{0,1\}^k$, such that $\mathrm{DDH}(s_{recv}, s_{sent}, \sigma_3) = 1$, $\mathrm{DDH}(s_{sent}, \mathsf{pk}_{s_{peer}}, \sigma_2) = 1$ and $\mathrm{DDH}(s_{recv}, \mathsf{pk}_{s_{actor}}, \sigma_1) = 1$, then S stores $(s_{actor}, s_{peer}, \mathcal{I}, s_{sent}, s_{recv}, \lambda)$ in table T.
   - Else, S chooses $\mu \in_R \{0,1\}^k$ and stores the entry $(s_{actor}, s_{peer}, \mathcal{I}, s_{sent}, s_{recv}, \mu)$ in T.

   Else if $s_{role} = \mathcal{R}$, then S does the following. If $V \notin G$, then the status of session s is set to `rejected`. Else, S sets the status of session s to `accepted`, returns g^κ to M, where κ denotes the last element of the entry $(s, s_{rand}, l_s, \mathsf{sk}_{s_{actor}}, \kappa)$ in table Q, and proceeds in a similar way as in the previous case.
5. When M makes a query of the form $\left(\sigma_1, \sigma_2, \sigma_3, \hat{P}_i, \hat{P}_j\right)$ to the random oracle for H_2, answer it as follows:
   - If $\left(\sigma_1, \sigma_2, \sigma_3, \hat{P}_i, \hat{P}_j, \lambda\right) \in L$ for some $\lambda \in \{0,1\}^k$, then S returns λ to M.
   - Else if there exist entries $\left(\hat{P}_i, \hat{P}_j, \mathcal{I}, U, V, \lambda\right)$ or $\left(\hat{P}_j, \hat{P}_i, \mathcal{R}, V, U, \lambda\right)$ in table T, for some $\lambda \in \{0,1\}^k$ and $U, V \in G$, such that $\mathrm{DDH}(V, U, \sigma_3) = 1$, $\mathrm{DDH}(V, \mathsf{pk}_{\hat{P}_i}, \sigma_1) = 1$ and $\mathrm{DDH}(U, \mathsf{pk}_{\hat{P}_j}, \sigma_2) = 1$, then S returns λ to M and stores the entry $\left(\sigma_1, \sigma_2, \sigma_3, \hat{P}_i, \hat{P}_j, \lambda\right)$ in table L.
   - Else, S chooses $\mu \in_R \{0,1\}^k$, returns it to M and stores the entry $\left(\sigma_1, \sigma_2, \sigma_3, \hat{P}_i, \hat{P}_j, \mu\right)$ in L.
6. $\mathsf{randomness}(s)$: If $s_{status} = \perp$, then S returns $\perp$. Otherwise, S returns s_{rand}.
7. $\mathsf{session\text{-}key}(s)$: If $s_{status} \neq \texttt{accepted}$, then S returns $\perp$. Otherwise, S answers this query by lookup in table T.
8. $\mathsf{test\text{-}session}(s)$: If $s \neq s^*$, then S aborts; otherwise S answers the query in the appropriate way.
9. $\mathsf{corrupt}(\hat{P})$: If $\hat{P} \notin \mathcal{P}$, then S returns $\perp$. Else if $\hat{P} = \hat{C}$, then S aborts. Else, S returns $\mathsf{sk}_{\hat{P}}$.
10. M outputs a guess: S aborts.

Analysis of event $DL \wedge K$

Similar to the analysis of the related event $DL \wedge K$ in the proof of [18, Proposition 7].

Event $T_O \wedge DL^c \wedge K$

Let s^* and s' denote the test session and the origin-session for the test session, respectively. We split event $Evt := T_O \wedge DL^c \wedge K$ into the following events $B_1, ..., B_3$ so that $Evt = B_1 \vee B_2 \vee B_3$:

1. B_1 : Evt occurs and $s^*_{peer} = s'_{actor}$.
2. B_2 : Evt occurs and $s^*_{peer} \neq s'_{actor}$ and M does not issue the queries $\mathsf{randomness}$ or $\mathsf{cr\text{-}create}$ to all sessions of s'_{actor} that were created prior to creation of the origin-session s' of s^*, including the origin-session itself, but may issue a $\mathsf{corrupt}(s^*_{peer})$ query.
3. B_3 : Evt occurs and $s^*_{peer} \neq s'_{actor}$ and M does not issue a $\mathsf{corrupt}(s^*_{peer})$ query, but may issue the queries $\mathsf{randomness}$ or $\mathsf{cr\text{-}create}$ to all session created prior to creation of the origin-session, including the origin-session s' itself.

Event B_1

Let the input to the GDH challenge be (X_0, Y_0). Suppose that event B_1 occurs with non-negligible probability. In this case S chooses long-term secret/public key pairs for all the honest parties and stores the associated long-term secret keys. Additionally S chooses two random values $m, n \in_R \{1, 2, ..., q_s\}$. The m'th activated session by adversary M will be called s^* and the n'th activated session will be called s'. Suppose further that $s^*_{actor} = \hat{A}, s^*_{peer} = \hat{B}$ and $s^*_{role} = \mathcal{I}$, w.l.o.g.. We now define S's responses to M's queries. S maintains tables Q, J, T and L, all of which are initially empty, as well as a variable ω initialized with 1.

1. $\mathsf{create}(\hat{A}, \mathcal{I}, \hat{B})$ or $\mathsf{cr\text{-}create}(\hat{A}, \mathcal{I}, str, \hat{B})$ to create session s^*: If create is issued, S chooses $s^*_{rand} \in_R \{0,1\}^k$. Else, S sets $s^*_{rand} \leftarrow str$. S (a) returns the message X_0, where (X_0, Y_0) is the GDH challenge, and (b) stores the entry $(s^*, s^*_{rand}, l_{s^*}, \mathsf{sk}_{\hat{A}}, *)$ in table Q, where $l_{s^*} = (s^*_{rand}, l_s)$ if there exists a previously created session s of user $s_{actor} = \hat{A}$ with an entry in table Q, and $l_{s^*} = s^*_{rand}$ if there no such session exists.
2. $\mathsf{create}(\hat{B}, r, \hat{Q})$ or $\mathsf{cr\text{-}create}(\hat{B}, r, str, \hat{Q})$ with $r \in \{\mathcal{I}, \mathcal{R}\}$ to create session s': If create is issued, S chooses $s'_{rand} \in_R \{0,1\}^k$. Else, S sets $s'_{rand} \leftarrow str$. S stores the entry $(s', s'_{rand}, l_{s'}, \mathsf{sk}_{\hat{B}}, *)$ in table Q, where $l_{s'} = (s'_{rand}, l_s)$ if there exists a previously created session s of user $s_{actor} = \hat{A}$ with an entry in table Q, and $l_{s'} = s'_{rand}$ if there no such session exists. If $r = \mathcal{I}$, then S returns message Y_0 to M, where (X_0, Y_0) is the GDH challenge. Else, $\star$ is returned.
3. $\mathsf{send}(\hat{B}, i, Z)$ with $(\hat{B}, i) = s'$: If $s'_{status} \neq \mathtt{active}$, then S returns $\perp$. Else if $s'_{role} = \mathcal{R}$ and $Z \in G$, then S returns message Y_0 to M, where (X_0, Y_0) is the GDH challenge, sets the status of session s' to $\mathtt{accepted}$, and proceeds as in the previous simulation for completing the session. Else, S proceeds as in the previous simulation.
4. $\mathsf{send}(\hat{A}, i, Y_0)$ with $(\hat{A}, i) = s^*$: S proceeds as in the previous simulation for completing the session.
5. Other $\mathsf{create}, \mathsf{cr\text{-}create}$ and send queries are answered as in the simulation relative to event $DL \wedge K$.
6. When M makes a query of the form $\left(\sigma_1, \sigma_2, \sigma_3, \hat{P}_i, \hat{P}_j\right)$ to the random oracle for H_2, answer it as follows:
   - If $\left\{\hat{P}_i, \hat{P}_j\right\} = \left\{\hat{A}, \hat{B}\right\}$, $\sigma_1 = Y_0^a$, $\sigma_2 = X_0^b$ and $\mathrm{DDH}(X_0, Y_0, \sigma_3) = 1$, then S aborts M and is successful by outputting $CDH(X_0, Y_0) = \sigma_3$.
   - Else if $\left(\sigma_1, \sigma_2, \sigma_3, \hat{P}_i, \hat{P}_j, \lambda\right) \in L$ for some $\lambda \in \{0,1\}^k$, then S returns λ to M.
   - Else if there exist entries $\left(\hat{P}_i, \hat{P}_j, \mathcal{I}, U, V, \lambda\right)$ or $\left(\hat{P}_j, \hat{P}_i, \mathcal{R}, V, U, \lambda\right)$ in table T, for some $\lambda \in \{0,1\}^k$ and $U, V \in G$, such that $\mathrm{DDH}(V, U, \sigma_3) = 1$, $\mathrm{DDH}(V, \mathsf{pk}_{\hat{P}_i}, \sigma_1) = 1$ and $\mathrm{DDH}(U, \mathsf{pk}_{\hat{P}_j}, \sigma_2) = 1$, then S returns λ to M and stores the entry $\left(\sigma_1, \sigma_2, \sigma_3, \hat{P}_i, \hat{P}_j, \lambda\right)$ in table L.
   - Else, S chooses $\mu \in_R \{0,1\}^k$, returns it to M and stores the entry $\left(\sigma_1, \sigma_2, \sigma_3, \hat{P}_i, \hat{P}_j, \mu\right)$ in L.
7. $\mathsf{randomness}(s)$: If $s_{status} = \perp$, then S returns $\perp$. Otherwise, S returns s_{rand}.

8. session-key(s): If $s_{status} \neq$ `accepted`, then S returns $\perp$. Otherwise, S answers this query by lookup in table T.
9. test-session(s): If $s \neq s^*$ or if s' is not the origin-session for session s^*, then S aborts; otherwise S answers the query in the appropriate way.
10. $H_1(r, h)$: If $r = l_{s^*}$ and $h = \mathsf{sk}_{\hat{A}}$ or if $r = l_{s'}$ and $h = \mathsf{sk}_{\hat{B}}$, then S aborts. Otherwise S simulates a random oracle as in the simulation relative to event $DL \wedge K$.
11. corrupt($\hat{P}$): If $\hat{P} \notin \mathcal{P}$, then S returns $\perp$. Else, S returns $\mathsf{sk}_{\hat{P}}$.
12. M outputs a guess: S aborts.

Analysis of event B_1
S's simulation of M's environment is perfect except with negligible probability. The probability that M selects s^* as the test-session and s' as the origin-session for the test-session is $\frac{1}{(q_s)^2}$. Assuming that this is indeed the case, S does not abort in Step 9. Under event DL^c, the adversary first issues a corrupt($\hat{P}$) query to party $\hat{P}$ before making an H_1 query that involves the long-term secret key of party $\hat{P}$. Freshness of the test session guarantees that the adversary can reveal/determine either l_{s^*} or $\mathsf{sk}_{\hat{A}}$, but not both. Similar for $l_{s'}$ and $\mathsf{sk}_{\hat{B}}$. Hence S does not abort in Step 10. Under event K, except with negligible probability of guessing $CDH(X_0, Y_0)$, S is successful as described in the first case of Step 6 and does not abort as in Step 12. Hence, if event B_1 occurs, then the success probability of S is given by $P(S) \geq \frac{1}{(q_s)^2} P(B_1)$.

Event B_2
Let the input to the GDH challenge be (X_0, Y_0). Suppose that event B_2 occurs with non-negligible probability. The simulation of S proceeds in the same way as for event B_1 with the following changes. S additionally keeps a history H of M's queries.

- randomness(s): If $s_{status} = \perp$, then S returns $\perp$. Else if $s = s'$ and there were queries (randomness or cr-create) to all previous sessions of the same user s'_{actor}, then S aborts. Else, S returns s_{rand}.
- $H_1(r, h)$: If $r = l_{s^*}$ and $h = \mathsf{sk}_{\hat{A}}$, then S aborts. Otherwise S simulates a random oracle as in the previous simulation.

Analysis of event B_2
Similar to the analyses of the related event B_2 in the proof of [18, Proposition 7] and event B_1.

Event B_3
Let the input to the GDH challenge be (X_0, B). Suppose that event B_3 occurs with non-negligible probability. In this case, S chooses one user $\hat{B} \in \mathcal{P}$ at random from the set $\mathcal{P}$ and sets its long-term public key to B. S chooses long-term secret/public key pairs for the remaining parties in $\mathcal{P}$ and stores the associated long-term secret keys. Additionally S chooses two random values $m, n \in_R \{1, 2, ..., q_s\}$. We denote the m'th activated session by adversary M by s^* and the n'th activated session by s'. Suppose further that $s^*_{actor} = \hat{A}, s^*_{peer} = \hat{B}$ and $s^*_{role} = \mathcal{I}$, w.l.o.g.. Algorithm S maintains tables Q, J, T and L, all of which are initially empty. S also maintains a variable ω initialized with 1.

1. create($\hat{A}, \mathcal{I}, \hat{B}$) or cr-create($\hat{A}, \mathcal{I}, str, \hat{B}$) to create session s^*: If create is issued, S chooses $s^*_{rand} \in_R \{0,1\}^k$. Else, S sets $s^*_{rand} \leftarrow str$. S (a) returns the message X_0, where (X_0, B) is the GDH challenge, and (b) stores the entry $(s^*, s^*_{rand}, l_{s^*}, \mathsf{sk}_{\hat{A}}, *)$ in table Q, where $l_{s^*} = (s^*_{rand}, l_s)$ if there exists a previously created session s of user $s_{actor} = \hat{A}$ with an entry in table Q, and $l_{s^*} = s^*_{rand}$ if there no such session exists.
2. create $\left(\hat{P}, r, \hat{Q}\right)$ to create session s: S checks whether $\hat{P} \in \mathcal{P}$, $\hat{Q} \in \mathcal{P}$, and $r \in \{\mathcal{I}, \mathcal{R}\}$. If one of the checks fails, then S returns $\perp$. Else, S initializes the session variables according to the protocol specification, and stores an entry of the form $(s, s_{rand}, l_s, \mathsf{sk}_{s_{actor}}, \kappa) \in (\mathcal{P} \times \mathbb{N}) \times \{0,1\}^k \times \{0,1\}^* \times (\mathbb{Z}_p \cup \{*\}) \times \mathbb{Z}_p$ in table Q as follows:
    - S chooses $s_{rand} \in_R \{0,1\}^k$ (i. e. the randomness of session s),

- S chooses $\kappa \in_R \mathbb{Z}_p$,
- if there is no entry $(s, s_{rand}, l_s, \mathsf{sk}_{s_{actor}}, \kappa)$ in table Q such that $s_{actor} = \hat{P}$, then S sets the value of l_s to s_{rand}, else S sets the value of l_s to $(s_{rand}, l_{s'})$, where s' is the previous session with $s'_{actor} = s_{actor}$ for which an entry in table Q has been made.
- if $s_{actor} \neq \hat{B}$, then S stores the entry $(s, s_{rand}, s_{data}, \mathsf{sk}_{s_{actor}}, \kappa)$ in Q, else S stores the entry $(s, s_{rand}, s_{data}, *, \kappa)$ in Q, and
- if $r = \mathcal{I}$, then S returns the Diffie-Hellman exponential g^κ to M, else S returns $\star$.

3. $\mathsf{cr\text{-}create}\left(\hat{P}, r, str, \hat{Q}\right)$ to create session s: S checks whether $\hat{P} \in \mathcal{P}$, $\hat{Q} \in \mathcal{P}$, and $r \in \{\mathcal{I}, \mathcal{R}\}$. If one of the checks fails, then S returns $\perp$. Else, S initializes the session variables according to the protocol specification, and stores an entry of the form $(s, s_{rand}, l_s, \mathsf{sk}_{s_{actor}}, \kappa) \in (\mathcal{P} \times \mathbb{N}) \times \{0,1\}^k \times \{0,1\}^* \times (\mathbb{Z}_p \cup \{*\}) \times \mathbb{Z}_p$ in table Q as follows:
   - if there is an entry (r_i, h_i, κ_i) in table J such that $r_i = (str, l_{s'})$, and $h_i = \mathsf{sk}_{\hat{P}}$, where s' is the previous session with $s'_{actor} = s_{actor}$ for which an entry in table Q has been made, then S sets $\omega \leftarrow \kappa_i$, else S chooses $\kappa \in_R \mathbb{Z}_p$ and sets $\omega \leftarrow \kappa$.
   - if $s_{actor} \neq \hat{B}$, then S stores the entry $(s, s_{rand}, r_i, \mathsf{sk}_{s_{actor}}, \omega)$ in Q, else S stores the entry $(s, s_{rand}, l_s, *, \omega)$ in Q with $l_s = (str, l_{s'})$,
   - if $r = \mathcal{I}$, then S returns the Diffie-Hellman exponential g^κ to M, else S returns $\star$.
4. S stores entries of the form $(r, h, \kappa) \in \{0,1\}^* \times \mathbb{Z}_p \times \mathbb{Z}_p$ in table J. When M makes a query of the form (r, h) to the random oracle for H_1, answer it as follows:
   - If $r = l_{s^*}$ and $h = \mathsf{sk}_{\hat{A}}$, then S aborts.
   - Else if $(r, h, \kappa) \in J$ for some $\kappa \in \mathbb{Z}_p$, then S returns κ to M.
   - Else if there exists an entry $(s, s_{rand}, l_s, \mathsf{sk}_{s_{actor}}, \kappa)$ in table Q with $l_s = r$ and $\mathsf{sk}_{s_{actor}} = h$, then S returns κ to M and stores the entry (r, h, κ) in table J.
   - Else, S chooses $\kappa \in_R \mathbb{Z}_p$, returns it to M and stores the entry (r, h, κ) in table J.
5. $\mathsf{send}(\hat{P}, i, V)$ to send message V to session $s = (\hat{P}, i)$: If $s_{status} \neq \texttt{active}$, then S returns $\perp$. Else if $s_{role} = \mathcal{I}$, then S does the following. If $V \notin G$, then the status of session s is set to `rejected`. Else, the status of session s is set to `accepted`, and
   - If there exists an entry $(s_{peer}, s_{actor}, \mathcal{R}, s_{recv}, s_{sent}, \lambda)$ in table T, then S stores the entry $(s_{actor}, s_{peer}, \mathcal{I}, s_{sent}, s_{recv}, \lambda)$ in table T.
   - Else if there exists an entry $(\sigma_1, \sigma_2, \sigma_3, s_{actor}, s_{peer}, \lambda)$ in table L, for some $\lambda \in \{0,1\}^k$, such that $\mathrm{DDH}(s_{recv}, s_{sent}, \sigma_3) = 1$, $\mathrm{DDH}(s_{sent}, \mathsf{pk}_{s_{peer}}, \sigma_2) = 1$ and $\mathrm{DDH}(s_{recv}, \mathsf{pk}_{s_{actor}}, \sigma_1) = 1$, then S stores $(s_{actor}, s_{peer}, \mathcal{I}, s_{sent}, s_{recv}, \lambda)$ in table T.
   - Else, S chooses $\mu \in_R \{0,1\}^k$ and stores the entry $(s_{actor}, s_{peer}, \mathcal{I}, s_{sent}, s_{recv}, \mu)$ in T.

   Else if $s_{role} = \mathcal{R}$, then S does the following. If $V \notin G$, then the status of session s is set to `rejected`. Else, S sets the status of session s to `accepted`, returns g^κ to M, where κ denotes the last element of the entry $(s, s_{rand}, l_s, \mathsf{sk}_{s_{actor}}, \kappa)$ in table Q, and proceeds in a similar way as in the previous case.
6. When M makes a query of the form $\left(\sigma_1, \sigma_2, \sigma_3, \hat{P}_i, \hat{P}_j\right)$ to the random oracle for H_2, answer it as follows:
   - If $\left\{\hat{P}_i, \hat{P}_j\right\} = \left\{\hat{A}, \hat{B}\right\}$, $\sigma_1 = A^\kappa$, $\mathrm{DDH}(X_0, B, \sigma_2) = 1$, and $\sigma_3 = X_0^\kappa$, where κ denotes the last element of the entry $(s', s'_{rand}, l_{s'}, \mathsf{sk}_{s'_{actor}}, \kappa)$ in table Q, then S aborts M and is successful by outputting $CDH(X_0, B) = \sigma_2$.
   - Else if $\left(\sigma_1, \sigma_2, \sigma_3, \hat{P}_i, \hat{P}_j, \lambda\right) \in L$ for some $\lambda \in \{0,1\}^k$, then S returns λ to M.
   - If $\left(\sigma_1, \sigma_2, \sigma_3, \hat{P}_i, \hat{P}_j, \lambda\right) \in L$ for some $\lambda \in \{0,1\}^k$, then S returns λ to M.
   - Else if there exist entries $\left(\hat{P}_i, \hat{P}_j, \mathcal{I}, U, V, \lambda\right)$ or $\left(\hat{P}_j, \hat{P}_i, \mathcal{R}, V, U, \lambda\right)$ in table T, for some $\lambda \in \{0,1\}^k$ and $U, V \in G$, such that $\mathrm{DDH}(V, U, \sigma_3) = 1$, $\mathrm{DDH}(V, \mathsf{pk}_{\hat{P}_i}, \sigma_1) = 1$ and

$\mathrm{DDH}(U, \mathsf{pk}_{\hat{P}_j}, \sigma_2) = 1$, then S returns λ to M and stores the entry $\left(\sigma_1, \sigma_2, \sigma_3, \hat{P}_i, \hat{P}_j, \lambda\right)$ in table L.

- Else, S chooses $\mu \in_R \{0,1\}^k$, returns it to M and stores the entry $\left(\sigma_1, \sigma_2, \sigma_3, \hat{P}_i, \hat{P}_j, \mu\right)$ in L.

7. $\mathsf{randomness}(s)$: If $s_{status} = \perp$, then S returns $\perp$. Otherwise, S returns s_{rand}.
8. $\mathsf{session\text{-}key}(s)$: If $s_{status} \neq \texttt{accepted}$, then S returns $\perp$. Otherwise, S answers this query by lookup in table T.
9. $\mathsf{test\text{-}session}(s)$: If $s \neq s^*$ or if s' is not the origin-session for session s^*, then S aborts; otherwise S answers the query in the appropriate way.
10. $\mathsf{corrupt}(\hat{P})$: If $\hat{P} \notin \mathcal{P}$, then S returns $\perp$. Else if $\hat{P} = \hat{B}$, then S aborts. Else, S returns $\mathsf{sk}_{\hat{P}}$.
11. M outputs a guess: S aborts.

Analysis of event B_3
Similar to the analysis of the related event B_3 in the proof of [18, Proposition 7].

Event $(T_O)^c \wedge DL^c \wedge K$
The simulation and analysis are very similar to the simulation and analysis related to event B_3. □