Basic Definitions, Constructions, and Proofs for Unilateral Authenticated Key Exchange

Doreen Riepel[1] and Paul Rösler[2]

[1] CISPA Helmholtz Center for Information Security, `riepel@cispa.de`
[2] FAU Erlangen-Nürnberg, `paul.roesler@fau.de`

March 17, 2025

Abstract. We give a compact game-based definition in pseudo-code for the key-indistinguishability of a simple class of *unilateral authenticated two-pass key exchange* protocols. Examples for this protocol class include a variant of *Signed Diffie-Hellman*, *Ntor*, and *Noise*'s *NK1* scheme. For the former two protocols, we provide reduction-based security proofs via sequences of games. The main purpose of our definition is to find a good balance between precision, simplicity, compactness, and strength of modeled security. For this reason, our security definitions require and our proofs cover only *forward security* but not *resilience against weak randomness sources*.

1 General Definitions

1.1 Notation

Symbol	Meaning
$[n]$	The set $\{1, \ldots, n\}$
`tru` / `fal`	Boolean values true and false
$[\![x]\!]$	Evaluates expression x to a Boolean value
$\leftarrow$ / $\rightarrow$	Assigns a constant expression or the output of a deterministic algorithm or oracle call
$\leftarrow_\$$ / $\rightarrow_\$$	Assigns a random value uniformly sampled from a finite set or the output of a probabilistic algorithm
$X \overset{\cup}{\leftarrow} x$	$X \leftarrow X \cup x$
$X[\cdot] \leftarrow x$	Assigns value x to all addresses of array X
$X \leftarrow 0^n$ / $\perp^n$ / $\emptyset^n$ / $\ldots$	Assigns an initial value to all n components of X
$(x, \cdot) \leftarrow y$ / $(x, \cdot) = y$	Assigns/compares the first value of tuple y to/with x and does not care about the second value
Game	Declares a security experiment
Oracle	Declares an oracle that the adversary can query during a security experiment
Internal Oracle	Declares an oracle that the only the challenger can query during a security experiment
Invoke X	Executes algorithm X
X^{O}	Algorithm X has oracle access to O (we omit the superscript when oracles are listed in the same figure)
Return x	Terminates a running algorithm or oracle with output x that is give to the calling instance or adversary, resp.
Stop with x	Terminates the running experiment with final output x
Require x	Ends the algorithm, resp. oracle, if expression $x =$ `fal`
$\mathbin{\\\\} \mathsf{G}_i$	Code line that is only executed in game i

1.2 Groups and Diffie-Hellman Problems

We let $\mathbb{G}$ be a group of prime order p with generator g and refer to $(\mathbb{G}, p, g)$ as a group specification. We will use this throughout the document, even though some elliptic curves used in practice, like `Curve25519`, are no prime-order groups. This is a typical assumption made in academic papers by the cryptography

Game St-CDH$_{\mathbb{G},p,g}(\mathcal{A})$	**Oracle** DDH(X, Y, Z)
$x, y \leftarrow_\$ \mathbb{Z}_p$	If $X = g^x$:
$Z \leftarrow_\$ \mathcal{A}(g^x, g^y)$	Return $[\![Z = Y^x]\!]$
Stop with $[\![Z = g^{xy}]\!]$	If $Y = g^y$:
	Return $[\![Z = X^y]\!]$
	Return 0

Fig. 1. Game St-CDH for a group specification $(\mathbb{G}, p, g)$.

community, however, we note that the given results should extend (with a little bit of extra work) to non prime-order elliptic curves as well.[3]

We use the strong Diffie-Hellman problem (St-CDH), a variant of the computational Diffie-Hellman problem that additionally provides the adversary access to a (fixed-base) decisional Diffie-Hellman oracle. The game is given in Figure 1 and we define the advantage of an adversary $\mathcal{A}$ in this game as $\mathrm{Adv}^{\text{st-cdh}}_{\mathbb{G},p,g}(\mathcal{A}) := \Pr[\text{St-CDH}_{\mathbb{G},p,g}(\mathcal{A}) = 1]$.

Remark 1. Compared to the original strong Diffie-Hellman problem, we allow the adversary to query the DDH oracle for both g^x and g^y. Hence, strictly speaking, our assumption is stronger. However, it maintains the flavor of a "fixed base" which makes it falsifiable, as opposed to a full "gap" oracle that allows arbitrary inputs.

We also give a corresponding multi-user variant of the problem in Figure 2. It will not only simplify our analysis but also give us tight reductions. In this game, the adversary can generate challenges g^{x_i} and g^{y_j} via oracles Gen_1 and Gen_2, where we denote the maximum number of queries to those oracles by q_1 and q_2, respectively. We also allow adaptive corruptions of elements output by Gen_1 and Gen_2 via oracles $\mathrm{Corrupt}_1$ and $\mathrm{Corrupt}_2$, respectively, which will leak the respective exponent. The goal is to compute the CDH solution for an uncorrupted pair of challenges.

Compared to the single-user definition, we do not ask the adversary to directly output the solution. Instead, we check via a flag win whether the adversary has queried a valid solution to its DDH oracle (lines 11-13). This is mainly to avoid book-keeping in later proofs and does not affect other aspects of the proof. (In particular, it does not affect the lemma below.) We define the advantage of $\mathcal{A}$ as $\mathrm{Adv}^{(q_1,q_2)\text{-st-cdh}}_{\mathbb{G},p,g}(\mathcal{A}) := \Pr[(q_1, q_2)\text{-St-CDH}_{\mathbb{G},p,g}(\mathcal{A}) = 1]$. The relation to the standard strong Diffie-Hellman problem is given in the following lemma.

Game (q_1, q_2)-St-CDH$_{\mathbb{G},p,g}(\mathcal{A})$	**Oracle** DDH(X, Y, Z)
00 win $\leftarrow 0$	11 If $\exists i : X = g^{x_i} \wedge Z = Y^{x_i}$:
01 $(n, m) \leftarrow 0^2$	12 If $\exists j : Y = g^{y_j} \wedge i \notin CR_1 \wedge j \notin CR_2$
02 $(CR_1, CR_2) \leftarrow \emptyset^2$	13 win $\leftarrow 1$
03 Invoke $\mathcal{A}$	14 Return 1
04 Stop with win	15 Else if $\exists j : Y = g^{y_j}$:
Oracle Gen_1 \\ at most q_1 queries	16 Return $[\![Z = X^{y_i}]\!]$
05 $n \leftarrow n + 1$	17 Return 0
06 $x_n \leftarrow_\$ \mathbb{Z}_p$	**Oracle** $\mathrm{Corrupt}_1(i)$
07 Return g^{x_n}	18 Require $i \in [n]$
Oracle Gen_2 \\ at most q_2 queries	19 $CR_1 \xleftarrow{\cup} \{i\}$
08 $m \leftarrow m + 1$	20 Return x_i
09 $y_m \leftarrow_\$ \mathbb{Z}_p$	**Oracle** $\mathrm{Corrupt}_2(j)$
10 Return g^{y_m}	21 Require $j \in [m]$
	22 $CR_2 \xleftarrow{\cup} \{j\}$
	23 Return y_j

Fig. 2. Game (q_1, q_2)-St-CDH for a group specification $(\mathbb{G}, p, g)$.

Lemma 1. *Let $(\mathbb{G}, p, g)$ be a group specification, q_1, q_2 be positive integers and $\mathcal{A}$ be an adversary against* (q_1, q_2)-St-CDH *security that issues q_{ddh} queries to* DDH. *Then there exists an adversary $\mathcal{B}$ against* St-CDH *such that*

$$\mathrm{Adv}^{(q_1,q_2)\text{-st-cdh}}_{\mathbb{G},p,g}(\mathcal{A}) \leq q_1 q_2 \cdot \mathrm{Adv}^{\text{st-cdh}}_{\mathbb{G},p,g}(\mathcal{B}),$$

[3] e.g., by adopting the syntax and bounds from Section 4.1 in https://eprint.iacr.org/2020/1499.pdf

where $\mathcal{B}$ makes at most q_{ddh} queries to its own DDH *oracle. If $\mathcal{A}$ does not have access to oracle* $\mathrm{Corrupt}_2$*, then we have*

$$\mathrm{Adv}^{(q_1,q_2)\text{-st-cdh}}_{\mathbb{G},p,g}(\mathcal{A}) \le q_1 \cdot \mathrm{Adv}^{\text{st-cdh}}_{\mathbb{G},p,g}(\mathcal{B}),$$

and analogously if the oracle $\mathrm{Corrupt}_1$ *is absent.*

1.3 Signatures

Syntax. A digital signature scheme specifies a public key space $\mathcal{PK}$, a secret key space $\mathcal{SK}$, a message space $\mathcal{M}$ and signature space $\mathcal{S}$ and defines the following algorithms.

- $\mathrm{SIG.gen} : \emptyset \to_\$ \mathcal{SK} \times \mathcal{PK}$ Signing and verification key generation
- $\mathrm{SIG.sig} : \mathcal{SK} \times \mathcal{M} \to_\$ \mathcal{S}$ Signing procedure
- $\mathrm{SIG.vfy} : \mathcal{PK} \times \mathcal{M} \times \mathcal{S} \to \{0,1\}$ Signature verification

Correctness. A signature scheme is (perfectly) correct if for all $(sk, pk) \leftarrow_\$ \mathrm{SIG.gen}$, all $m \in \mathcal{M}$ we have:

$$\Pr[\mathrm{SIG.vfy}(pk, m, \mathrm{SIG.sig}(sk, m)) = 1] = 1.$$

Entropy of Key Generation. We require that public keys generated by SIG.gen are not "predictable". More formally, we say that key generation has γ_{SIG} bits of entropy if for all (possibly unbounded) adversaries $\mathcal{A}$ we have $\Pr[pk = pk' \mid pk' \leftarrow \mathcal{A}; (\cdot, pk) \leftarrow_\$ \mathrm{SIG.gen}] \le 2^{-\gamma_{\mathrm{SIG}}}$.

Security. In Figure 3, we give the game for strong unforgeability under chosen message attacks (SUF-CMA). In this game, the adversary has to produce a valid message-signature pair (m, σ), while having access to a signing oracle. We require *strong* unforgeability, meaning that the adversary is also allowed to come up with a new signature σ' for a message m that it previously asked to sign.

Similar to above, we also give a multi-user version of that game (q-SUF-CMA) in Figure 4, where q is the maximum number of users. The game supports adaptive corruptions via an oracle Corrupt. We define the advantage of an adversary $\mathcal{A}$ as $\mathrm{Adv}^{\text{suf-cma}}_{\mathrm{SIG}}(\mathcal{A}) := \Pr[\text{SUF-CMA}_{\mathrm{SIG}}(\mathcal{A}) = 1]$ and $\mathrm{Adv}^{q\text{-suf-cma}}_{\mathrm{SIG}}(\mathcal{A}) := \Pr[q\text{-SUF-CMA}_{\mathrm{SIG}}(\mathcal{A}) = 1]$, respectively.

It is well-known that single-user security implies multi-user security (with a loss linear in the number of users), as captured by the following lemma.

Game $\text{SUF-CMA}_{\mathrm{SIG}}(\mathcal{A})$
$Q \leftarrow \emptyset$
$(sk, pk) \leftarrow_\$ \mathrm{SIG.gen}$
$(m, \sigma) \leftarrow_\$ \mathcal{A}(pk)$
If $(m, \sigma) \notin Q$
    Stop with $\mathrm{SIG.vfy}(pk, m, \sigma)$
Stop with 0

Oracle $\mathrm{Sign}(m)$
$\sigma \leftarrow_\$ \mathrm{SIG.sig}(sk, m)$
$Q \xleftarrow{\cup} \{(m, \sigma)\}$
Return c

Fig. 3. Game SUF-CMA for signature scheme SIG.

Game $q\text{-SUF-CMA}_{\mathrm{SIG}}(\mathcal{A})$
$n \leftarrow 0$
$(Q, CR) \leftarrow \emptyset^2$
$(j, m, \sigma) \leftarrow_\$ \mathcal{A}$
If $j \notin CR \wedge (j, m, \sigma) \notin Q$
    Stop with $\mathrm{SIG.vfy}(pk_j, m, \sigma)$
Stop with 0

Oracle Gen   \\ at most q queries
$n \leftarrow n + 1$
$(sk_n, pk_n) \leftarrow_\$ \mathrm{SIG.gen}$
Return pk_n

Oracle $\mathrm{Sign}(j, m)$
Require $j \in [n]$
$\sigma \leftarrow_\$ \mathrm{SIG.sig}(sk_j, m)$
$Q \xleftarrow{\cup} \{(j, m, \sigma)\}$
Return c

Oracle $\mathrm{Corrupt}(j)$
Require $j \in [n]$
$CR \xleftarrow{\cup} \{j\}$
Return sk_j

Fig. 4. Game q-SUF-CMA for signature scheme SIG.

Lemma 2. *Let* SIG *be a signature scheme,* q *be a positive integer and* $\mathcal{A}$ *be an adversary against* q-SUF-CMA *security of* SIG *that issues* q_{sig} *queries to* Sign*. Then there exists an adversary* $\mathcal{B}$ *against* SUF-CMA *security of* SIG *such that*

$$\mathrm{Adv}_{\mathrm{SIG}}^{q\text{-suf-cma}}(\mathcal{A}) \leq q \cdot \mathrm{Adv}_{\mathrm{SIG}}^{\text{suf-cma}}(\mathcal{B}),$$

where $\mathcal{B}$ *makes at most* q_{sig} *queries to its own* Sign *oracle.*

2 Unilateral Authenticated Two-Pass Key Exchange

To use a minimal, non-trivial class of protocols for our modeling task, we consider *unilateral authenticated two-pass key exchange* $\mathrm{KE} = (\mathrm{KE.gen}, \mathrm{KE.init}, \mathrm{KE.resp}, \mathrm{KE.recv})$.

Syntax. A two-pass key exchange protocol specifies a public key space $\mathcal{PK}$, a secret key space $\mathcal{SK}$, a state space $\mathcal{ST}$, ciphertext spaces $\mathcal{C}_1$ and $\mathcal{C}_2$, and a session key space $\mathcal{K}$. Then the algorithms are defined as follows.

- $\mathrm{KE.gen} : \emptyset \rightarrow_\$ \mathcal{SK} \times \mathcal{PK}$ Long-term key generation for responders
- $\mathrm{KE.init} : \mathcal{PK} \rightarrow_\$ \mathcal{ST} \times \mathcal{C}_1$ First protocol step executed by the initiator
- $\mathrm{KE.resp} : \mathcal{SK} \times \mathcal{C}_1 \rightarrow_\$ \mathcal{K} \times \mathcal{C}_2$ Second protocol step executed by the responder
- $\mathrm{KE.recv} : \mathcal{ST} \times \mathcal{C}_2 \rightarrow \mathcal{K}$ Final receiving step executed by the initiator

Correctness. Using game $\mathrm{CORR}_{\mathrm{KE}}$ from Figure 5 we define scheme KE correct iff

$$\Pr[\mathrm{CORR}_{\mathrm{KE}}(\mathcal{A}) = 0] = 1.$$

Intuitively, this game models that if the communication between an initiator and a responder is honestly forwarded (via oracles Init, Respond, and Receive), then the keys computed by the responder and the initiator should be nontrivial (i.e., $k \neq \perp$, see line 20) and equal (line 11).

Thus, as a clarification for members of the key-exchange research community: initiator and responder are considered to be "*partnered*" iff they processed the same transcript and the responder is the initiator's *intended partner* (lines 07, 19, 21, and 11). Thereby we avoid adding an additional layer of arbitrary identifiers for parties and simply use cryptographic public keys as the identifiers of responders—adding a PKI or other methods on top of this is orthogonal. (This means that the so called "*session identifier*" is the concatenation of the transcript and the responder's public key.)

Game $\mathrm{CORR}_{\mathrm{KE}}(\mathcal{A})$
00 $(n, m) \leftarrow 0^2$
01 $(P[\cdot], I[\cdot], R[\cdot]) \leftarrow \perp^3$
02 Invoke $\mathcal{A}$
03 Stop with 0

Oracle $\mathrm{Init}(pk)$
04 $n \leftarrow n + 1$
05 $(st_n, c) \leftarrow_\$ \mathrm{KE.init}(pk)$
06 If $\exists j \in [m] : pk = pk_j$:
07   $P[n] \leftarrow j$; $I[n] \leftarrow c$
08 Return c

Oracle $\mathrm{Receive}(i, c)$
09 Require $i \in [n]$
10 $k \leftarrow \mathrm{KE.recv}(st_i, c)$
11 If $R[P[i], i, c] \notin \{k, \perp\}$:
12   Stop with 1
13 Return k

Oracle Gen
14 $m \leftarrow m + 1$
15 $(sk_m, pk_m) \leftarrow_\$ \mathrm{KE.gen}$
16 Return pk_m

Oracle $\mathrm{Respond}(j, c)$
17 Require $j \in [m]$
18 $(k, c') \leftarrow_\$ \mathrm{KE.resp}(sk_j, c)$
19 If $\exists i \in [n] : P[i] = j \wedge I[i] = c$:
20   If $k = \perp$: Stop with 1
21   $R[j, i, c'] \leftarrow k$
22 Return (k, c')

Fig. 5. Game CORR for key exchange KE.

Game $\mathrm{IND}^b_{\mathrm{KE}}(\mathcal{A})$
00 $(n, m) \leftarrow 0^2$
01 $(P[\cdot], I[\cdot]) \leftarrow \bot^2$
02 $(Q, \mathit{ICH}, \mathit{RCH}, \mathit{XP}, \mathit{CR}, R[\cdot]) \leftarrow \emptyset^6$
03 $b' \leftarrow_{\$} \mathcal{A}$
04 Stop with b'

Oracle $\mathrm{Init}(pk)$
05 $n \leftarrow n + 1$
06 $(st_n, c) \leftarrow_{\$} \mathrm{KE.init}(pk)$
07 If $\exists j \in [m] : pk = pk_j$:
08 $\quad P[n] \leftarrow j;\ I[n] \leftarrow c$
09 Return c

Oracle $\mathrm{Receive}(i, c, ch)$
10 Require $i \in [n] \setminus Q$
11 $Q \xleftarrow{\cup} \{i\}$
12 If $c \in R[P[i], i]$: Return
13 $k \leftarrow \mathrm{KE.recv}(st_i, c)$
14 If $ch \wedge i \notin \mathit{XP} \wedge P[i] \in [m] \setminus \mathit{CR} \wedge k \neq \bot$:
15 $\quad$ If $b = 1$: $k \leftarrow_{\$} \mathcal{K}$
16 $\quad \mathit{ICH} \xleftarrow{\cup} \{i\}$
17 $\quad \mathit{RCH} \xleftarrow{\cup} \{P[i]\}$
18 Return k

Oracle Gen
19 $m \leftarrow m + 1$
20 $(sk_m, pk_m) \leftarrow_{\$} \mathrm{KE.gen}$
21 Return pk_m

Oracle $\mathrm{Respond}(j, c, ch)$
22 Require $j \in [m]$
23 $(k, c') \leftarrow_{\$} \mathrm{KE.resp}(sk_j, c)$
24 If $\exists i \in [n] : P[i] = j \wedge I[i] = c$:
25 $\quad R[j, i] \xleftarrow{\cup} \{c'\}$
26 $\quad$ If $ch \wedge i \notin \mathit{XP}$:
27 $\quad\quad$ If $b = 1$: $k \leftarrow_{\$} \mathcal{K}$
28 $\quad\quad \mathit{ICH} \xleftarrow{\cup} \{i\}$
29 Return (k, c')

Oracle $\mathrm{Corrupt}(j)$
30 Require $j \in [m] \setminus \mathit{RCH}$
31 $\mathit{CR} \xleftarrow{\cup} \{j\}$
32 Return sk_j

Oracle $\mathrm{Expose}(i)$
33 Require $i \in [n] \setminus \mathit{ICH}$
34 $\mathit{XP} \xleftarrow{\cup} \{i\}$
35 Return st_i

Fig. 6. Game IND for key exchange KE. Line 17 is only necessary for implicitly authenticated protocols like ntor. For explicitly authenticated protocols (e.g., based on signatures), line 17 can be ignored, which strengthens the adversary.

2.1 Game-Based Security

We define security via games $\mathrm{IND}^b_{\mathrm{KE}}$ from Figure 6, where adversary $\mathcal{A}$ can query the following oracles:

- $\mathrm{Gen} \to pk$ generates a new responder key pair and outputs the public key pk.
- $\mathrm{Init}(pk) \to c$ initiates a session with intended partner pk, which results in ciphertext c.
- $\mathrm{Respond}(j, c, ch) \to (k, c')$ lets responder j with pk_j respond to incoming ciphertext c, which yields output ciphertext c'. If $\mathcal{A}$ wants to be challenged on the resulting output key via $ch = 1$ and the incoming ciphertext was output by an unexposed initiator whose intended partner is pk_j (line 24), then the output key is either the real key or a randomly sampled one, depending on challenge bit b (line 27); otherwise, the real key is output.
- $\mathrm{Receive}(i, c, ch) \to k/\bot$ lets initiator i receive ciphertext c. If c is a response of the intended partner to i's initial ciphertext, nothing is output (line 12; since, by correctness, the same key was output by oracle Respond already); otherwise real key k is output, unless the adversary queried a challenge via $ch = 1$, the initiator state is unexposed, the responder secret key is uncorrupted, and challenge bit $b = 1$, in which case a random key is output (lines 14-15).
- $\mathrm{Expose}(i) \to st_i$ exposes the local state of initiator i.
- $\mathrm{Corrupt}(j) \to sk_j$ corrupts the long-term secret key of responder j.

The adversary terminates with a guess b' such that the advantage in winning the game is defined as

$$\mathrm{Adv}^{\mathrm{ind}}_{\mathrm{KE}}(\mathcal{A}) := \left|\Pr[\mathrm{IND}^0_{\mathrm{KE}}(\mathcal{A}) = 1] - \Pr[\mathrm{IND}^1_{\mathrm{KE}}(\mathcal{A}) = 1]\right| .$$

Intuition for Modeled Security. As a clarification for members of the key-exchange research community: the common idea of oracle "Reveal" is covered by oracles $\mathrm{Respond}(j, c, ch = 0)$ and $\mathrm{Receive}(i, c, ch = 0)$, respectively, and the idea of oracle "Test" is covered by oracles $\mathrm{Respond}(j, c, ch = 1)$ and $\mathrm{Receive}(i, c, ch = 1)$, respectively. By allowing arbitrary public key inputs pk in oracle $\mathrm{Init}(pk)$, we do not only cover corruption of long-term keys but also maliciously generated long-term keys.

We refrain from modeling weak randomness or leakage of responders' ephemeral secrets, since we consider sessions with weak randomness (entirely) insecure and we do not see realistic scenarios in which the responder's ephemeral key material is leaked.

Thus, intuitively, the keys computed by initiators and responders are required to be secure under the following conditions:

1. The responder's algorithm KE.resp outputs a secure key for an honestly received ciphertext (Figure 6 line 24), unless the honest initiator's ephemeral state is ever exposed (see lines 26, 28, 33-34).
2. Algorithm KE.recv outputs a secure key for adversarially crafted ciphertexts (line 12), unless the intended responder's long-term key is ever corrupted (lines 14, 17, 30-31), or the receiving initiator's ephemeral state is ever exposed (lines 14, 16, 33-34).
   Note that this property is sub-optimal and tailored to the considered construction: a stronger notion of forward security would require security if the responder's long-term key was corrupted only *before* the initiator received the adversarially crafted ciphertext. Therefore, line 17 is only necessary for implicitly authenticated protocols. For explicitly authenticated protocols (e.g., based on signatures), line 17 can be ignored, which strengthens the adversary. In this document, we will only use the stronger definition.

We refer the interested reader to two systematization of knowledge papers on definitions of key exchange by Poettering et al.[4] and Brzuska et al.[5].

3 Constructions

In this section, we recall the signed DH key exchange and simplified NTOR protocol from the problem set.[6] We also provide pen-and-paper proofs of security.

3.1 Signed Diffie-Hellman Key Exchange

In Figure 7, we give the signed Diffie-Hellman key exchange, short KE_2, using a signature scheme SIG and a hash function $\mathrm{Hash} : \mathbb{G}^3 \to \mathcal{K}$ that will be modeled as a random oracle.[7] We include (parts of) the transcript in the key derivation, see also Remark 5.

```
Proc KE_2.init(pk)
x ←$ Z_p
st ← (pk, x)
Return (st, c = g^x)

Proc KE_2.recv(st, c' = (g^y, σ))
(pk, x) ← st
If SIG.vfy(pk, (g^x, g^y), σ) = 0:
   Return ⊥
k ← Hash(g^x, g^y, (g^y)^x)
Return k

Proc KE_2.gen
(sk, pk) ←$ SIG.gen
Return (sk, pk)

Proc KE_2.resp(sk, c = g^x)
y ←$ Z_p
σ ←$ SIG.sig(sk, (c, g^y))
k ← Hash(c, g^y, c^y)
Return (k, c' = (g^y, σ))
```

Fig. 7. Signed Diffie-Hellman key exchange.

Correctness and Security. If the signature scheme is correct and the communication between initiator and responder is forwarded honestly, both compute the same session key $k = \mathrm{Hash}(g^x, g^y, g^{xy})$.

Further, we show that the protocol is secure if the signature scheme is strongly unforgeable and the strong computational Diffie-Hellman assumption holds. We first give a tight bound with respect to the multi-user assumptions.

Theorem 1. *Let* KE_2 *be the protocol from Figure 7, using a signature scheme* SIG *and a group specification* $(\mathbb{G}, p, g)$ *and* Hash *being modeled as a random oracle* H. *Let* $\mathcal{A}$ *be an adversary against* IND *security of* KE_2 *that makes at most* q_{gen} *queries to* Gen, q_{init} *queries to* Init, q_{resp} *queries to* Respond

[4] https://eprint.iacr.org/2021/305.pdf
[5] https://eprint.iacr.org/2024/1215.pdf
[6] https://github.com/charlie-j/fm-crypto-lib/
[7] We explicitly write $\mathbb{G}^3$ for the domain and will assume the adversary only queries the random oracle on (valid) group elements. Arbitrary strings can also be captured, but these queries are not "useful", so we ignore them in our modeling.

and q_{H} *queries to* H*. Then there exist adversaries* $\mathcal{B}_1$ *against* q_{gen}-SUF-CMA *security of* SIG *and* $\mathcal{B}_2$ *against* $(q_{\mathrm{init}}, q_{\mathrm{resp}})$-St-CDH *such that*

$$\mathrm{Adv}^{\mathrm{ind}}_{\mathrm{KE}_2}(\mathcal{A}) \le 2 \cdot \mathrm{Adv}^{q_{\mathrm{gen}}\text{-suf-cma}}_{\mathrm{SIG}}(\mathcal{B}_1) + \mathrm{Adv}^{(q_{\mathrm{init}}, q_{\mathrm{resp}})\text{-st-cdh}}_{\mathbb{G},p,g}(\mathcal{B}_2) + 2 \cdot q_{\mathrm{init}} q_{\mathrm{gen}} \cdot 2^{-\gamma_{\mathrm{SIG}}} + \frac{2q_{\mathrm{H}}(q_{\mathrm{init}} + q_{\mathrm{resp}})}{p},$$

where γ_{SIG} *is the key generation entropy of* SIG*.* $\mathcal{B}_1$ *makes* q_{resp} *queries to its* Sign *oracle.* $\mathcal{B}_2$ *makes* q_{H} *queries to* DDH*.*

We prove the theorem below. We note that the last term is a proof artifact and could be avoided, see Remark 3. We now derive the respective bound from standard single-user assumptions, using Lemmas 1 and 2 and observing that $\mathcal{B}_2$ does not require access to $\mathrm{Corrupt}_2$.

Corollary 1. *Let* KE_2 *be the protocol from Figure 7, using a signature scheme* SIG *and a group specification* $(\mathbb{G}, p, g)$ *and* Hash *being modeled as a random oracle* H*. Let* $\mathcal{A}$ *be an adversary against* IND *security of* KE_2 *that makes at most* q_{gen} *queries to* Gen*,* q_{init} *queries to* Init*,* q_{resp} *queries to* Respond *and* q_{H} *queries to* H*. Then there exist adversaries* $\mathcal{B}_1$ *against* SUF-CMA *security of* SIG *and* $\mathcal{B}_2$ *against* St-CDH *such that*

$$\mathrm{Adv}^{\mathrm{ind}}_{\mathrm{KE}_2}(\mathcal{A}) \le 2q_{\mathrm{gen}} \cdot \mathrm{Adv}^{\mathrm{suf\text{-}cma}}_{\mathrm{SIG}}(\mathcal{B}_1) + q_{\mathrm{init}} \cdot \mathrm{Adv}^{\mathrm{st\text{-}cdh}}_{\mathbb{G},p,g}(\mathcal{B}_2) + 2 \cdot q_{\mathrm{init}} q_{\mathrm{gen}} \cdot 2^{-\gamma_{\mathrm{SIG}}} + \frac{2q_{\mathrm{H}}(q_{\mathrm{init}} + q_{\mathrm{resp}})}{p},$$

where γ_{SIG} *is the key generation entropy of* SIG*.* $\mathcal{B}_1$ *makes at most* q_{resp} *queries to its* Sign *oracle.* $\mathcal{B}_2$ *makes* q_{H} *queries to* DDH*.*

Proof (of Theorem 1). Let $\mathcal{A}$ be an adversary against IND security of KE_2. We prove the theorem via the sequence of games given in Figure 8.

Game G_0*.* This is the original IND game for KE_2. We have

$$\mathrm{Adv}^{\mathrm{ind}}_{\mathrm{KE}_2}(\mathcal{A}) = \left|\Pr\left[\mathsf{G}^0_0(\mathcal{A}) \Rightarrow 1\right] - \Pr\left[\mathsf{G}^1_0(\mathcal{A}) \Rightarrow 1\right]\right|.$$

Game G_1*.* In this game, we handle the case that the adversary queries a public key pk to Init and later the same public key is output by Gen. We do so via setting a flag bad_1 in Gen after the key pair is generated and let the game stop directly if bad_1 is set. Since the two games are identical-until-bad, we have

$$\left|\Pr\left[\mathsf{G}^b_0(\mathcal{A}) \Rightarrow 1\right] - \Pr\left[\mathsf{G}^b_1(\mathcal{A}) \Rightarrow 1\right]\right| \le \Pr[\mathrm{bad}_1].$$

To see why this is an issue, let $pk^\star$ be a public key chosen by $\mathcal{A}$ and queried to Init. Then assume Gen computes (sk_m, pk_m) such that $pk_m = pk^\star$ and $\mathcal{A}$ corrupts that party. Using sk_m, it then computes a valid signature and challenges the initiator's session via Receive. Since the variable $P[n]$ is not set during Init, the challenge is allowed and $\mathcal{A}$ can win.

To bound bad_1, note that for each query to Gen, bad_1 is set with probability at most $n \cdot 2^{-\gamma_{\mathrm{SIG}}}$, where γ_{SIG} is the entropy of key generation and n is the current counter value for Init queries. Since $n \le q_{\mathrm{init}}$, we can the difference by taking a union bound over all Gen queries. Hence, we get

$$\Pr[\mathrm{bad}_1] \le q_{\mathrm{gen}} q_{\mathrm{init}} \cdot 2^{-\gamma_{\mathrm{SIG}}}.$$

Remark 2. We do not take into account collisions among key pairs output by Gen since this is captured within the multi-user definition of SUF-CMA. Note however that this means there could exist more than one index j such that line 11 evaluates to true. Technically, we would have to make a choice which index to pick, but we omit it for simplicity. Fortunately, no matter which index we chose in this case, it would not affect the reduction below. However, formal verification tools need to be more specific about this choice.

Games G_0^b-G_3^b
$(n, m) \leftarrow 0^2$
$(P[\cdot], I[\cdot], H[\cdot]) \leftarrow \perp^3$
$(Q, \mathit{ICH}, \mathit{RCH}, \mathit{XP}, \mathit{CR}, R[\cdot]) \leftarrow \emptyset^6$
$H'[\cdot] \leftarrow \perp$ $\quad\quad /\!\!/ \mathsf{G}_3^b$
$(\mathrm{bad}_1, \mathrm{bad}_2) \leftarrow 0^2$ $\quad\quad /\!\!/ \mathsf{G}_1^b$-$\mathsf{G}_3^b$
$b' \leftarrow_\$ \mathcal{A}$
Stop with b'

Oracle Init(pk)
$n \leftarrow n + 1$
$x \leftarrow_\$ \mathbb{Z}_p$
$c \leftarrow g^x$
$st_n \leftarrow (pk, x)$
If $\exists j \in [m] : pk = pk_j$:
  $P[n] \leftarrow j$; $I[n] \leftarrow c$
Return c

Oracle Receive(i, c, ch)
Require $i \in [n] \setminus Q$
$Q \xleftarrow{\cup} \{i\}$
If $c \in R[P[i], i]$: Return
$(pk, x) \leftarrow st_i$
$(h, \sigma) \leftarrow c$
If SIG.vfy$(pk, (g^x, h), \sigma) = 0$:
  $k \leftarrow \perp$
Else:
  $k \leftarrow \mathrm{H}(g^x, h, h^x)$ $\quad\quad /\!\!/ \mathsf{G}_0^b$-$\mathsf{G}_2^b$
  $k \leftarrow \mathrm{H}'(g^x, h)$ $\quad\quad /\!\!/ \mathsf{G}_3^b$
If $ch \wedge i \notin \mathit{XP} \wedge P[i] \notin \mathit{CR} \wedge k \neq \perp$:
  $\mathrm{bad}_2 \leftarrow 1$; Stop $\quad\quad /\!\!/ \mathsf{G}_2^b$-$\mathsf{G}_3^b$
  If $b = 1$: $k \leftarrow_\$ \mathcal{K}$
  $\mathit{ICH} \xleftarrow{\cup} \{i\}$
Return k

Internal Random Oracle $\mathrm{H}'(X, Y)$ $/\!\!/ \mathsf{G}_3^b$
If $H'[X, Y] \neq \perp$: Return $H'[X, Y]$
$k \leftarrow_\$ \mathcal{K}$
$H'[X, Y] \leftarrow k$
Return k

Oracle Gen
$m \leftarrow m + 1$
$(sk_m, pk_m) \leftarrow_\$$ SIG.gen
If $\exists i \in [n] : st_i = (pk_m, \cdot) \wedge P[i] = \perp$: $\quad\quad /\!\!/ \mathsf{G}_1^b$-$\mathsf{G}_3^b$
  $\mathrm{bad}_1 \leftarrow 1$; Stop $\quad\quad /\!\!/ \mathsf{G}_1^b$-$\mathsf{G}_3^b$
Return pk_m

Oracle Respond(j, c, ch)
Require $j \in [m]$
$y \leftarrow_\$ \mathbb{Z}_p$
$h \leftarrow g^y$
$\sigma \leftarrow_\$$ SIG.sig$(sk_j, (c, h))$
$c' \leftarrow (h, \sigma)$
$k \leftarrow \mathrm{H}(c, g^y, c^y)$ $\quad\quad /\!\!/ \mathsf{G}_0^b$-$\mathsf{G}_2^b$
If $\exists i \in [n] : I[i] = c$: $\quad\quad /\!\!/ \mathsf{G}_3^b$
  $k \leftarrow \mathrm{H}'(c, g^y)$ $\quad\quad /\!\!/ \mathsf{G}_3^b$
Else: $\quad\quad /\!\!/ \mathsf{G}_3^b$
  $k \leftarrow \mathrm{H}(c, g^y, c^y)$ $\quad\quad /\!\!/ \mathsf{G}_3^b$
If $\exists i \in [n] : P[i] = j \wedge I[i] = c$:
  $R[j, i] \xleftarrow{\cup} \{c'\}$
  If $ch \wedge i \notin \mathit{XP}$:
    If $b = 1$: $k \leftarrow_\$ \mathcal{K}$
    $\mathit{ICH} \xleftarrow{\cup} \{i\}$
Return (k, c')

Oracle Corrupt(j)
Require $j \in [m]$
$\mathit{CR} \xleftarrow{\cup} \{j\}$
Return sk_j

Oracle Expose(i)
Require $i \in [n] \setminus \mathit{ICH}$
$\mathit{XP} \xleftarrow{\cup} \{i\}$
Return st_i

Random Oracle $\mathrm{H}(X, Y, Z)$
If $H[X, Y, Z] \neq \perp$: Return $H[X, Y, Z]$
If $\exists i \in [n] : I[i] = X$: $\quad\quad /\!\!/ \mathsf{G}_3^b$
  $(\cdot, x) \leftarrow st_i$ $\quad\quad /\!\!/ \mathsf{G}_3^b$
  If $Z = Y^x$: $\quad\quad /\!\!/ \mathsf{G}_3^b$
    Return $\mathrm{H}'(X, Y)$ $\quad\quad /\!\!/ \mathsf{G}_3^b$
$k \leftarrow_\$ \mathcal{K}$
$H[X, Y, Z] \leftarrow k$
Return k

Fig. 8. Games for the proof of Theorem 1.

Game G_2. We introduce a flag bad_2 which is set to 1 in oracle Receive when the adversary asks for a challenge for an unexposed instance i, where the intended responder $P[i]$ is uncorrupted and a valid session key (i.e., $k \neq \perp$) has been computed. This means that Receive got as input a valid signature σ that was not previously output by oracle Respond. If bad_2 is set to 1, the game stops. We have

$$\left|\Pr\left[\mathsf{G}_1^b(\mathcal{A}) \Rightarrow 1\right] - \Pr\left[\mathsf{G}_2^b(\mathcal{A}) \Rightarrow 1\right]\right| \leq \Pr[\mathrm{bad}_2].$$

We now bound the probability of bad_2 by constructing an adversary $\mathcal{B}_{1,b}$ against q_{gen}-SUF-CMA security of SIG. Adversary $\mathcal{B}_{1,b}$ is given in Figure 9, where we denote $\mathcal{B}_{1,b}$'s key generation oracle by Gen$'$ and its corruption oracle by Corrupt$'$ to differentiate from $\mathcal{A}$'s oracles. We highlight those lines blue, where the simulation is nontrivial, e.g., $\mathcal{B}_{1,b}$ needs to call its own oracles.

Since $\mathcal{B}_{1,b}$ is an adversary in the multi-user game, the simulation is rather straightforward. To generate public keys, it queries its oracle Gen$'$. To sign messages, it queries its oracle Sign. To answer corruption queries, it queries its oracle Corrupt$'$. Finally, we observe that if bad_2 is set in G_2^b, then there exists an unexposed instance i that has no partnered responder session, where the intended responder $P[i]$ is uncorrupted and a valid session key has been computed. The latter means that the signature verification was successful and since there is no partnered session, the pair $((g^x, h), \sigma)$ is new (for party $P[i]$) and constitutes a valid forgery in $\mathcal{B}_{1,b}$'s game. Hence,

$$\Pr[\mathrm{bad}_2] \leq \mathrm{Adv}_{\mathrm{SIG}}^{q_{\mathrm{gen}}\text{-suf-cma}}(\mathcal{B}_{1,b}).$$

Adversary $\mathcal{B}_{1,b}^{\text{Gen}',\text{Sign},\text{Corrupt}'}$
$(n, m) \leftarrow 0^2$
$(P[\cdot], I[\cdot], H[\cdot]) \leftarrow \perp^3$
$(Q, ICH, RCH, XP, CR, R[\cdot]) \leftarrow \emptyset^6$
$b' \leftarrow_\$ \mathcal{A}$
Stop

Oracle $\text{Init}(pk)$
$n \leftarrow n + 1$
$x \leftarrow_\$ \mathbb{Z}_p$
$c \leftarrow g^x$
$st_n \leftarrow (pk, x)$
If $\exists j \in [m] : pk = pk_j$:
  $P[n] \leftarrow j$; $I[n] \leftarrow c$
Return c

Oracle $\text{Receive}(i, c, ch)$
Require $i \in [n] \setminus Q$
$Q \xleftarrow{\cup} \{i\}$
If $c \in R[P[i], i]$: Return
$(pk, x) \leftarrow st_i$
$(h, \sigma) \leftarrow c$
If $\text{SIG.vfy}(pk, (g^x, h), \sigma) = 0$
  $k \leftarrow \perp$
Else
  $k \leftarrow \text{H}(g^x, h, h^x)$
If $ch \wedge i \notin XP \wedge P[i] \notin CR \wedge k \neq \perp$:
  Stop with $(P[i], (g^x, h), \sigma)$
  If $b = 1$: $k \leftarrow_\$ \mathcal{K}$
  $ICH \xleftarrow{\cup} \{i\}$
Return k

Oracle Gen
$m \leftarrow m + 1$
$pk_m \leftarrow \text{Gen}'$
If $\exists i \in [n] : st_i = (pk_m, \cdot) \wedge P[i] = \perp$:
  Stop
Return pk_m

Oracle $\text{Respond}(j, c, ch)$
Require $j \in [m]$
$y \leftarrow_\$ \mathbb{Z}_p$
$\sigma \leftarrow \text{Sign}(j, (c, g^y))$
$k \leftarrow \text{H}(c, g^y, c^y)$
$c' \leftarrow (g^y, \sigma)$
If $\exists i \in [n] : P[i] = j \wedge I[i] = c$:
  $R[j, i] \xleftarrow{\cup} \{c'\}$
  If $ch \wedge i \notin XP$:
    If $b = 1$: $k \leftarrow_\$ \mathcal{K}$
    $ICH \xleftarrow{\cup} \{i\}$
Return (k, c')

Oracle $\text{Corrupt}(j)$
Require $j \in [m]$
$sk_j \leftarrow \text{Corrupt}'(j)$
$CR \xleftarrow{\cup} \{j\}$
Return sk_j

Oracle $\text{Expose}(i)$
Require $i \in [n] \setminus ICH$
$XP \xleftarrow{\cup} \{i\}$
Return st_i

Random Oracle $\text{H}(X, Y, Z)$
If $H[X, Y, Z] \neq \perp$: Return $H[X, Y, Z]$
$k \leftarrow_\$ \mathcal{K}$
$H[X, Y, Z] \leftarrow k$
Return k

Fig. 9. Adversary $\mathcal{B}_{1,b}$ against q_{gen}-SUF-CMA.

Game G_3. This game prepares for the final reduction. We introduce an internal random oracle H' which is used to simulate queries to $\text{H}(X, Y, Z)$, where $X = g^x$ was previously output by oracle Init and $Z = Y^x$. This oracle is called during Respond and Receive to keep partnered sessions consistent, as well as in the random oracle itself to keep H and H' consistent.

On first sight, these changes seem only conceptual. However, there is a small inconsistency we have to take into account. Namely, the adversary can notice a difference if it queries H on (X, Y, Z) where X is only *later* output by Init and $Z = Y^x$, in which case the game adds an additional entry to H. Similarly, if $\mathcal{A}$ queries H on (X, Y, Z) after X was output by Init, but *before* Y was output by Respond. Since x, y are chosen uniformly at random and there are at most q_{H} queries to H, we can bound the probability by

$$\left|\Pr\left[\mathsf{G}_2^b(\mathcal{A}) \Rightarrow 1\right] - \Pr\left[\mathsf{G}_3^b(\mathcal{A}) \Rightarrow 1\right]\right| \leq \frac{q_{\text{H}}(q_{\text{init}} + q_{\text{resp}})}{p}.$$

Remark 3. This upper bound is rather conservative and could be avoided when sampling all x_i and y_i at the beginning of the game instead of on the fly. Intuitively, this means that the adversary breaks CDH before seeing the challenge, which is why the reduction below can then also break CDH. Hence, we could also check for this when x_i and y_i are sampled, leading to a larger number of DDH queries in the later reduction. We omit these additional steps and simply add the above term which is sufficiently small for large p.

Finally, we construct an adversary $\mathcal{B}_2$ such that

$$\left|\Pr\left[\mathsf{G}_3^0(\mathcal{A}) \Rightarrow 1\right] - \Pr\left[\mathsf{G}_3^1(\mathcal{A}) \Rightarrow 1\right]\right| \leq \text{Adv}_{\mathbb{G},p,g}^{(q_{\text{init}}, q_{\text{resp}})\text{-st-cdh}}(\mathcal{B}_2).$$

Note that the only place where the two games differ is the challenge in oracle Respond, when queried on $(j, c, 1)$, where there exists an initiator session with intended partner j that has output c and this instance is unexposed. Also, as long as $\mathcal{A}$ does not query the random oracle on the respective input, the

Adversary $\mathcal{B}_2^{\mathrm{Gen}_1,\mathrm{Gen}_2,\mathrm{Corrupt}_1,\mathrm{DDH}}$
$(n, m) \leftarrow 0^2$
$(P[\cdot], I[\cdot], H[\cdot], H'[\cdot]) \leftarrow \perp^4$
$(Q, \mathit{ICH}, \mathit{RCH}, \mathit{XP}, \mathit{CR}, R[\cdot]) \leftarrow \emptyset^6$
$b' \leftarrow_\$ \mathcal{A}$
Stop

Oracle Init(pk)
$n \leftarrow n + 1$
$c \leftarrow \mathrm{Gen}_1$
$st_n \leftarrow (pk, \perp)$
If $\exists j \in [m] : pk = pk_j$:
  $P[n] \leftarrow j$; $I[n] \leftarrow c$
Return c

Oracle Receive(i, c, ch)
Require $i \in [n] \setminus Q$
$Q \xleftarrow{\cup} \{i\}$
If $c \in R[P[i], i]$: Return
$(pk, x) \leftarrow st_i$
$(h, \sigma) \leftarrow c$
If SIG.vfy$(pk, (g^x, h), \sigma) = 0$
  $k \leftarrow \perp$
Else:
  $k \leftarrow \mathrm{H}'(g^x, h)$
If $ch \wedge i \notin \mathit{XP} \wedge P[i] \notin \mathit{CR} \wedge k \neq \perp$:
  Stop
Return k

Oracle Expose(i)
Require $i \in [n] \setminus \mathit{ICH}$
$\mathit{XP} \xleftarrow{\cup} \{i\}$
$(pk, \cdot) \leftarrow st_i$
$x \leftarrow \mathrm{Corrupt}_1(i)$
Return (pk, x)

Internal Random Oracle $\mathrm{H}'(X, Y)$
If $H'[X, Y] \neq \perp$: Return $H'[X, Y]$
$k \leftarrow_\$ \mathcal{K}$
$H'[X, Y] \leftarrow k$
Return k

Oracle Gen
$m \leftarrow m + 1$
$(sk_m, pk_m) \leftarrow_\$ \mathrm{SIG.gen}$
If $\exists i \in [n] : st_i = (pk_m, \cdot) \wedge P[i] = \perp$:
  Stop
Return pk_m

Oracle Respond(j, c, ch)
Require $j \in [m]$
If $\exists i \in [n] : P[i] = j \wedge I[i] = c$:
  $h \leftarrow \mathrm{Gen}_2$
Else:
  $y \leftarrow_\$ \mathbb{Z}_p$
  $h \leftarrow g^y$
$\sigma \leftarrow_\$ \mathrm{SIG.sig}(sk_j, (c, h))$
If $\exists i \in [n] : P[i] = j \wedge I[i] = c$:
  $k \leftarrow \mathrm{H}'(c, h)$
Else:
  $k \leftarrow \mathrm{H}(c, h, c^y)$
$c' \leftarrow (h, \sigma)$
If $\exists i \in [n] : P[i] = j \wedge I[i] = c$:
  $R[j, i] \xleftarrow{\cup} \{c'\}$
  If $ch \wedge i \notin \mathit{XP}$:
    $\mathit{ICH} \xleftarrow{\cup} \{i\}$
Return (k, c')

Oracle Corrupt(j)
Require $j \in [m]$
$\mathit{CR} \xleftarrow{\cup} \{j\}$
Return sk_j

Random Oracle $\mathrm{H}(X, Y, Z)$
If $H[X, Y, Z] \neq \perp$: Return $H[X, Y, Z]$
If $\exists i \in [n] : I[i] = X \wedge \mathrm{DDH}(X, Y, Z) = 1$
  Return $\mathrm{H}'(X, Y)$
$k \leftarrow_\$ \mathcal{K}$
$H[X, Y, Z] \leftarrow k$
Return k

Fig. 10. Adversary $\mathcal{B}_2$ against $(q_{\mathrm{gen}}, q_{\mathrm{init}})$-St-CDH.

two games proceed identically. Therefore, we will show that if adversary $\mathcal{A}$ can distinguish the real session key from a uniformly random one, we can construct an adversary $\mathcal{B}_2$ winning the $(q_{\mathrm{init}}, q_{\mathrm{resp}})$-St-CDH game.

Adversary $\mathcal{B}_2$ is given in Figure 10 and proceeds as follows. For each of $\mathcal{A}$'s queries to Init, it queries Gen_1. Since it does not know the exponent x_n, it simply sets that part of the state to $\perp$. If $\mathcal{A}$ asks to expose the state, $\mathcal{B}_2$ can obtain it via its $\mathrm{Corrupt}_1$ oracle. When $\mathcal{A}$ queries Respond and there exists a partnered initiator instance, then $\mathcal{B}_2$ queries its Gen_2 oracle and uses the internal random oracle H' to compute the session key, without knowing Z. Otherwise, it simply samples y itself since this will not be a challenge, which allows it to honestly compute the session key. Finally, we look at the random oracle queries. If $\mathcal{A}$ queries H on (X, Y, Z) such that X was output by an initiator instance and the query constitutes a valid Diffie-Hellman tuple, which $\mathcal{B}_2$ can check via its oracle DDH, then $\mathcal{B}_2$ internally queries H'. Recall that in the $(q_{\mathrm{init}}, q_{\mathrm{resp}})$-St-CDH game, the winning condition is evaluated via flag win. Hence, if the initiator instance was not exposed and Y was output by the responder, then $\mathcal{B}_2$ wins. Note that the simulation is also perfect for exposed instances. Further, $\mathcal{B}_2$ does not require access to $\mathrm{Corrupt}_2$.

This concludes the description of $\mathcal{B}_2$. The theorem now follows from collecting the probabilities and folding adversaries $\mathcal{B}_{1,0}$ and $\mathcal{B}_{1,1}$ into a single adversary $\mathcal{B}_1$. □

Remark 4. Since we rely on strong unforgeability, it is actually not necessary to add instance i to set *ICH* in Receive. For this observe that an initiator session (that has no partnered responder session) can never be challenged, unless a signature is forged. This holds independent of whether the initiator's state has been exposed or not, hence we could also allow exposure in this case.

Remark 5. Depending on what exactly is included in the key derivation hash, we get different security bounds and require different assumptions.

If we did not include g^x, g^y and only set $k = \mathrm{Hash}(g^{xy})$, the bounds would be less tight since each time the random oracle is queried on a value Z, we would not know which instance this query belonged to. More specifically, while the advantage bound could be the same, adversary $\mathcal{B}_2$ would have to make about $q_{\mathrm{init}} q_{\mathrm{resp}} q_{\mathrm{H}}$ queries to DDH in the worst case. Alternatively, we can possibly achieve a non-tight advantage bound from standard CDH (without DDH oracle) when guessing the random oracle query for which CDH will be solved as well as the two instances involved, which would lead to a loss of $q_{\mathrm{init}} q_{\mathrm{resp}} q_{\mathrm{H}}$ and which further complicates the proof.

If we additionally included σ in the hash, existential unforgeability would be sufficient to prove the theorem. In that case, we would need to rely on strong CDH whenever a "rerandomized" signature is given as input to Receive. (In this case, the addition to set *ICH* in Receive is necessary.) We note that this is a consequence of the way we define partnering. Alternative partnering definitions may ignore signatures, leading to a different theorem statement.

3.2 Simplified NTOR

In Figure 11, we give the simplified NTOR protocol, short KE_5, using a hash function $\mathrm{Hash} : \mathbb{G}^5 \to \mathcal{K}^2$ that we will model as a random oracle.

Proc $\mathrm{KE}_5.\mathrm{init}(pk)$
00 $x \leftarrow_\$ \mathbb{Z}_p$
01 $st \leftarrow (pk, x)$
02 Return $(st, c = g^x)$

Proc $\mathrm{KE}_5.\mathrm{recv}(st, c' = (g^y, auth))$
03 $(pk, x) \leftarrow st$
04 $(auth', k) \leftarrow \mathrm{Hash}(g^x, g^y, pk, (g^y)^x, pk^x)$
05 If $auth' \neq auth$:
06 $\quad$ Return $\perp$
07 Return k

Proc $\mathrm{KE}_5.\mathrm{gen}$
08 $sk \leftarrow_\$ \mathbb{Z}_p$
09 $pk \leftarrow g^{sk}$
10 Return (sk, pk)

Proc $\mathrm{KE}_5.\mathrm{resp}(sk, c = g^x)$
11 $y \leftarrow_\$ \mathbb{Z}_p$
12 $(auth, k) \leftarrow \mathrm{Hash}(c, g^y, g^{sk}, c^y, c^{sk})$
13 Return $(k, c' = (g^y, auth))$

Fig. 11. Simplified NTOR protocol.

Remark 6. In the original problem set, the parties first compute a key mk via $\mathrm{Hash}(g^x, g^y, g^{sk}, g^{xy}, g^{xsk})$ and then use mk to derive $auth$ and k. We simplify the protocol to derive those values directly (see lines 04, 12).

Correctness and Security. It is easy to see that the protocol is correct since both parties compute $(auth, k) = \mathrm{Hash}(g^x, g^y, g^{sk}, g^{xy}, g^{xsk})$. We state the security below, first from multi-user assumptions and then from (standard) single-user assumptions, using Lemma 1 and observing that adversary $\mathcal{B}_2$ does not require access to oracle $\mathrm{Corrupt}_2$.

Theorem 2. *Let* KE_5 *be the protocol from Figure 11, using a group specification* $(\mathbb{G}, p, g)$ *and* Hash *being modeled as a random oracle* H. *Let* $\mathcal{A}$ *be an adversary against* IND *security of* KE_5 *that makes at most* q_{gen} *queries to* Gen, q_{init} *queries to* Init, q_{resp} *queries to* Respond *and* q_{H} *queries to* H. *Then there exist adversaries* $\mathcal{B}_1$ *against* $(q_{\mathrm{gen}}, q_{\mathrm{init}})$-St-CDH *and* $\mathcal{B}_2$ *against* $(q_{\mathrm{init}}, q_{\mathrm{resp}})$-St-CDH *such that*

$$\mathrm{Adv}^{\mathrm{ind}}_{\mathrm{KE}_5}(\mathcal{A}) \leq 2 \cdot \mathrm{Adv}^{(q_{\mathrm{gen}}, q_{\mathrm{init}})\text{-st-cdh}}_{\mathbb{G},p,g}(\mathcal{B}_1) + \mathrm{Adv}^{(q_{\mathrm{init}}, q_{\mathrm{resp}})\text{-st-cdh}}_{\mathbb{G},p,g}(\mathcal{B}_2) + \frac{2 q_{\mathrm{init}}}{|\mathcal{K}|} + \frac{2 q_{\mathrm{gen}} q_{\mathrm{init}}}{p} + \frac{2 q_{\mathrm{H}} (q_{\mathrm{init}} + q_{\mathrm{resp}})}{p}.$$

$\mathcal{B}_1$ *makes* q_{gen} *queries to its own* Gen *oracle and* q_{resp} *queries to its* Sign *oracle.* $\mathcal{B}_2$ *makes* q_{H} *queries to* DDH.

Again we want to note that the last term is a proof artifact and could be avoided, see Remark 3 in the previous proof.

Corollary 2. *Let* KE_5 *be the protocol from Figure 11, using a group specification* $(\mathbb{G}, p, g)$ *and* Hash *being modeled as a random oracle* H. *Let* $\mathcal{A}$ *be an adversary against* IND *security of* KE_5 *that makes at*

Games G_0^b-G_4^b

$(n, m) \leftarrow 0^2$
$(P[\cdot], I[\cdot], H[\cdot]) \leftarrow \perp^3$
$(Q, ICH, RCH, XP, CR, R[\cdot]) \leftarrow \emptyset^6$
$(H_1[\cdot], H_2[\cdot], H_3[\cdot], RS[\cdot]) \leftarrow \perp^4$ $\quad /\!\!/ \mathsf{G}_2^b$-$\mathsf{G}_4^b$
$(\mathrm{bad}_1, \mathrm{bad}_2) \leftarrow 0^2$ $\quad /\!\!/ \mathsf{G}_1^b$-$\mathsf{G}_4^b$
$b' \leftarrow_\$ \mathcal{A}$
Stop with b'

Oracle Init(pk)

$n \leftarrow n + 1$
$x \leftarrow_\$ \mathbb{Z}_p$
$c \leftarrow g^x$
$st_n \leftarrow (pk, x)$
If $\exists j \in [m] : pk = pk_j$:
  $P[n] \leftarrow j$; $I[n] \leftarrow c$
Return c

Oracle Receive(i, c, ch)

Require $i \in [n] \setminus Q$
$Q \xleftarrow{\cup} \{i\}$
If $c \in R[P[i], i]$: Return
$(pk, x) \leftarrow st_i$
$(h, auth) \leftarrow c$
If $H_3[g^x, h, pk] = \perp$: Return $\perp$ $\quad /\!\!/ \mathsf{G}_3^b$-$\mathsf{G}_4^b$
$(auth', k) \leftarrow \mathrm{H}(g^x, h, pk, h^x, pk^x)$ $\quad /\!\!/ \mathsf{G}_0^b$-$\mathsf{G}_1^b$
$(auth', k) \leftarrow \mathrm{H}_3(g^x, h, pk)$ $\quad /\!\!/ \mathsf{G}_2^b$-$\mathsf{G}_4^b$
If $auth' \neq auth$:
  $k \leftarrow \perp$
If $ch \wedge i \notin XP \wedge P[i] \notin CR \wedge k \neq \perp$:
  $\mathrm{bad}_2 \leftarrow 1$; Stop $\quad /\!\!/ \mathsf{G}_4^b$
  If $b = 1$: $k \leftarrow_\$ \mathcal{K}$
  $ICH \xleftarrow{\cup} \{i\}$
Return k

Oracle Expose(i)

Require $i \in [n] \setminus ICH$
$XP \xleftarrow{\cup} \{i\}$
Return st_i

Internal Random Oracle $\mathrm{H}_i(X, Y, pk)$ $\quad /\!\!/ i \in [3]$, G_2-G_4

If $H_i[X, Y, pk] \neq \perp$: Return $H_i[X, Y, pk]$
$(auth, k) \leftarrow_\$ \mathcal{K}^2$
$H_i[X, Y, pk] \leftarrow (auth, k)$
Return $(auth, k)$

Oracle Gen

$m \leftarrow m + 1$
$sk_m \leftarrow_\$ \mathbb{Z}_p$
$pk_m \leftarrow g^{sk_m}$
If $\exists i \in [n] : st_i = (pk_m, \cdot) \wedge P[i] = \perp$: $\quad /\!\!/ \mathsf{G}_1^b$-$\mathsf{G}_4^b$
  $\mathrm{bad}_1 \leftarrow 1$; Stop $\quad /\!\!/ \mathsf{G}_1^b$-$\mathsf{G}_4^b$
Return pk_m

Oracle Respond(j, c, ch)

Require $j \in [m]$
$y \leftarrow_\$ \mathbb{Z}_p$
$h \leftarrow g^y$
$(auth, k) \leftarrow \mathrm{H}(c, h, pk_j, c^y, c^{sk_j})$ $\quad /\!\!/ \mathsf{G}_0^b$-$\mathsf{G}_1^b$
If $\exists i \in [n] : P[i] = j \wedge I[i] = c$: $\quad /\!\!/ \mathsf{G}_2^b$-$\mathsf{G}_4^b$
  $(auth, k) \leftarrow \mathrm{H}_1(c, h, pk_j)$ $\quad /\!\!/ \mathsf{G}_2^b$-$\mathsf{G}_4^b$
Else: $\quad /\!\!/ \mathsf{G}_2^b$-$\mathsf{G}_4^b$
  $RS[h] \leftarrow y$ $\quad /\!\!/ \mathsf{G}_2^b$-$\mathsf{G}_4^b$
  $(auth, k) \leftarrow \mathrm{H}_2(c, h, pk_j)$ $\quad /\!\!/ \mathsf{G}_2^b$-$\mathsf{G}_4^b$
$c' \leftarrow (h, auth)$
If $\exists i \in [n] : P[i] = j \wedge I[i] = c$:
  $R[j, i] \xleftarrow{\cup} \{c'\}$
  If $ch \wedge i \notin XP$:
    If $b = 1$: $k \leftarrow_\$ \mathcal{K}$
    $ICH \xleftarrow{\cup} \{i\}$
Return (k, c')

Oracle Corrupt(j)

Require $j \in [m]$
$CR \xleftarrow{\cup} \{j\}$
Return sk_j

Random Oracle $\mathrm{H}(X, Y, pk, Z_1, Z_2)$

If $H[X, Y, pk, Z_1, Z_2] \neq \perp$:
  Return $H[X, Y, pk, Z_1, Z_2]$
If $\exists j \in [m] : pk = pk_j \wedge Z_2 = X^{sk_j}$: $\quad /\!\!/ \mathsf{G}_2^b$-$\mathsf{G}_4^b$
  If $\exists i \in [n] : I[i] = X$: $\quad /\!\!/ \mathsf{G}_2^b$-$\mathsf{G}_4^b$
    $(\cdot, x) \leftarrow st_i$ $\quad /\!\!/ \mathsf{G}_2^b$-$\mathsf{G}_4^b$
    If $Z_1 = Y^x$: $\quad /\!\!/ \mathsf{G}_2^b$-$\mathsf{G}_4^b$
      If $Y \in R[j, i]$: Return $\mathrm{H}_1(X, Y, pk)$ $\quad /\!\!/ \mathsf{G}_2^b$-$\mathsf{G}_4^b$
      Return $\mathrm{H}_3(X, Y, pk)$ $\quad /\!\!/ \mathsf{G}_2^b$-$\mathsf{G}_4^b$
  Else if $RS[Y] \neq \perp \wedge Z_1 = X^{RS[Y]}$: $\quad /\!\!/ \mathsf{G}_2^b$-$\mathsf{G}_4^b$
    Return $\mathrm{H}_2(X, Y, pk)$ $\quad /\!\!/ \mathsf{G}_2^b$-$\mathsf{G}_4^b$
$(auth, k) \leftarrow_\$ \mathcal{K}^2$
$H[X, Y, pk, Z_1, Z_2] \leftarrow (auth, k)$
Return $(auth, k)$

Fig. 12. Games for the proof of Theorem 2.

most q_{gen} queries to Gen*, q_{init} queries to* Init*, q_{resp} queries to* Respond *and q_{H} queries to* H*. Then there exist adversaries $\mathcal{B}_1$ against* SUF-CMA *security of* SIG *and $\mathcal{B}_2$ against* St-CDH *such that*

$$\mathrm{Adv}^{\mathrm{ind}}_{\mathrm{KE}_5}(\mathcal{A}) \leq 2q_{\mathrm{gen}}q_{\mathrm{init}} \cdot \mathrm{Adv}^{\text{st-cdh}}_{\mathbb{G},p,g}(\mathcal{B}_1) + q_{\mathrm{init}} \cdot \mathrm{Adv}^{\text{st-cdh}}_{\mathbb{G},p,g}(\mathcal{B}_2) + \frac{2q_{\mathrm{init}}}{|\mathcal{K}|} + \frac{2q_{\mathrm{gen}}q_{\mathrm{init}}}{p} + \frac{2q_{\mathrm{H}}(q_{\mathrm{init}} + q_{\mathrm{resp}})}{p}.$$

$\mathcal{B}_2$ makes q_{H} queries to DDH.

Proof (of Theorem 2). Let $\mathcal{A}$ be an adversary against IND security of KE_5. We prove the theorem via the sequence of games given in Figure 12.

Game G_0. This is the original IND game for KE_5. We have

$$\mathrm{Adv}^{\mathrm{ind}}_{\mathrm{KE}_5}(\mathcal{A}) = \left|\Pr\left[\mathsf{G}_0^0(\mathcal{A}) \Rightarrow 1\right] - \Pr\left[\mathsf{G}_0^1(\mathcal{A}) \Rightarrow 1\right]\right|.$$

Game G_1. Similar to the previous proof, we first handle the case that the adversary queries a public key pk to Init and later the same public key is output by Gen. We set flag bad_1 in Gen and since secret keys are chosen uniformly from $\mathbb{Z}_p$, we obtain

$$\left|\Pr\left[\mathsf{G}_0^b(\mathcal{A}) \Rightarrow 1\right] - \Pr\left[\mathsf{G}_1^b(\mathcal{A}) \Rightarrow 1\right]\right| \leq \Pr[\mathrm{bad}_1] \leq \frac{q_{\mathrm{gen}}q_{\mathrm{init}}}{p}.$$

Game G_2. We now introduce three internal random oracles H_1, H_2 and H_3, which are used to simulate $\mathrm{H}(X, Y, pk, Z_1, Z_2)$ queries where Z_1 and Z_2 are valid Diffie-Hellman solutions. We distinguish the following three cases:

– H_1 is used when $X = g^x$ was previously output by oracle Init and Y was output by Respond for a corresponding partnered session.
– H_2 is used when Y was output by Respond, but no partnered initiator session exists. Note that this case will never lead to a challenge query, but we need to keep H_2 consistent with H since the adversary can (potentially) compute valid Z_1 and Z_2.
– H_3 is used when $X = g^x$ was previously output by oracle Init, but Y was chosen by the adversary. We will use this random oracle to handle challenges to the Receive oracle.

As in the previous proof, we need to consider the event that the adversary could have queried the random oracle on X or Y before they were output by the game. Hence,

$$\left|\Pr\left[\mathsf{G}_1^b(\mathcal{A}) \Rightarrow 1\right] - \Pr\left[\mathsf{G}_2^b(\mathcal{A}) \Rightarrow 1\right]\right| \leq \frac{q_{\mathrm{H}}(q_{\mathrm{init}} + q_{\mathrm{resp}})}{p}.$$

Game G_3. In this game, we change the simulation of Receive. Whenever the oracle is queried on a ciphertext h such that the entry $H_3[g^x, h, pk]$ is $\perp$, where (pk, x) is the instance's state, the game directly returns $\perp$. This is to prepare for the next game hop, namely, to ensure that the adversary can only provide a valid tag *auth* by first querying the random oracle (and thus breaking CDH). We claim that

$$\left|\Pr\left[\mathsf{G}_2^b(\mathcal{A}) \Rightarrow 1\right] - \Pr\left[\mathsf{G}_3^b(\mathcal{A}) \Rightarrow 1\right]\right| \leq \frac{q_{\mathrm{init}}}{|\mathcal{K}|}.$$

To see this, first note that whenever there exists a partnered responder session, the entry $H_3[g^x, h, pk]$ will be defined. Hence, we only care about sessions that do not have a partner and, more specifically, we need to consider the case that Receive outputs $\perp$ in game G_3^b, but a valid key k in game G_2^b. For the latter, the check $auth' = auth$ must succeed. For a particular query to Receive, the probability that the check would succeed even if the entry $H_3[g^x, h, pk]$ is not defined prior to the oracle call is $1/|\mathcal{K}|$. Union bound over all queries to Receive and observing that there can be at most q_{init} queries yields the above bound.

Game G_4. In this game, we introduce a flag bad_2 which is set to 1 in oracle Receive when the adversary asks for a challenge for an unexposed instance i, where the intended responder $P[i]$ is uncorrupted and a valid session key (i.e., $k \neq \perp$) has been computed. This means that Receive got as input a valid tag *auth* that was not previously output by oracle Respond. If bad is set to 1, the game stops. Since the two games are identical-until-bad, we have

$$\left|\Pr\left[\mathsf{G}_3^b(\mathcal{A}) \Rightarrow 1\right] - \Pr\left[\mathsf{G}_4^b(\mathcal{A}) \Rightarrow 1\right]\right| \leq \Pr[\mathrm{bad}_2].$$

We bound the probability of bad_2 by constructing an adversary $\mathcal{B}_{1,b}$ against $(q_{\mathrm{gen}}, q_{\mathrm{init}})$-St-CDH.

Adversary $\mathcal{B}_{1,b}$ is given in Figure 13 and proceeds as follows. It uses its Gen_1 oracle to generate long-term keys and its $\mathrm{Corrupt}_1$ oracle to answer $\mathcal{A}$'s queries to Corrupt. It simulates queries to Init using its Gen_2 oracle and its $\mathrm{Corrupt}_2$ oracle to answer $\mathcal{A}$'s queries to Expose. The main difficulty is the simulation of the random oracle. Fortunately, the checks on Z_1 and Z_2 can be translated into DDH queries by observing that the check on Z_2 always contains a long-term public key of a responder and that the check on Z_1 always contains an element output by Init. Finally, recall that the $(q_{\mathrm{gen}}, q_{\mathrm{init}})$-St-CDH game determines whether $\mathcal{B}_{1,b}$ wins via flag win which is set when DDH is queried on the right Z_2 and both X and Y have not been corrupted. This is exactly the condition for bad_2. Hence, whenever bad_2 is set, there must have been such a query to H and $\mathcal{B}_{1,b}$ wins. We get

$$\Pr[\mathrm{bad}_2] \leq \mathrm{Adv}_{\mathbb{G},p,g}^{(q_{\mathrm{gen}},q_{\mathrm{init}})\text{-st-cdh}}(\mathcal{B}_{1,b}).$$

Finally, we construct an adversary $\mathcal{B}_2$ such that

$$\left|\Pr\left[\mathsf{G}_4^0(\mathcal{A}) \Rightarrow 1\right] - \Pr\left[\mathsf{G}_4^1(\mathcal{A}) \Rightarrow 1\right]\right| \leq \mathrm{Adv}_{\mathbb{G},p,g}^{(q_{\mathrm{init}},q_{\mathrm{resp}})\text{-st-cdh}}(\mathcal{B}_2).$$

Adversary $\mathcal{B}_{1,b}^{\mathrm{Gen}_1,\mathrm{Gen}_2,\mathrm{Corrupt}_1,\mathrm{Corrupt}_2,\mathrm{DDH}}$
$(n, m) \leftarrow 0^2$
$(P[\cdot], I[\cdot], H[\cdot]) \leftarrow \bot^3$
$(Q, ICH, RCH, XP, CR, R[\cdot]) \leftarrow \emptyset^6$
$(H_1[\cdot], H_2[\cdot], H_3[\cdot], RS[\cdot]) \leftarrow \bot^4$
$(\mathrm{bad}_1, \mathrm{bad}_2) \leftarrow 0^2$
$b' \leftarrow_\$ \mathcal{A}$
Stop

Oracle Init(pk)
$n \leftarrow n + 1$
$x \leftarrow_\$ \mathbb{Z}_p$
$c \leftarrow \mathrm{Gen}_2$
$st_n \leftarrow (pk, \bot)$
If $\exists j \in [m] : pk = pk_j$:
  $P[n] \leftarrow j$; $I[n] \leftarrow c$
Return c

Oracle Receive(i, c, ch)
Require $i \in [n] \setminus Q$
$Q \xleftarrow{\cup} \{i\}$
If $c \in R[P[i], i]$: Return
$(pk, x) \leftarrow st_i$
$(h, auth) \leftarrow c$
If $H_3[g^x, h, pk] = \bot$: Return $\bot$
$(auth', k) \leftarrow \mathrm{H}_3(g^x, h, pk)$
If $auth' \neq auth$:
  $k \leftarrow \bot$
If $ch \wedge i \notin XP \wedge P[i] \notin CR \wedge k \neq \bot$:
  Stop
Return k

Oracle Expose(i)
Require $i \in [n] \setminus ICH$
$XP \xleftarrow{\cup} \{i\}$
$(pk, \cdot) \leftarrow st_i$
$x \leftarrow \mathrm{Corrupt}_2(i)$
Return (pk, x)

Internal Random Oracle $\mathrm{H}_i(X, Y, pk)$
If $H_i[X, Y, pk] \neq \bot$: Return $H_i[X, Y, pk]$
$(auth, k) \leftarrow_\$ \mathcal{K}^2$
$H_i[X, Y, pk] \leftarrow (auth, k)$
Return $(auth, k)$

Oracle Gen
$m \leftarrow m + 1$
$pk_m \leftarrow \mathrm{Gen}_1$
If $\exists i \in [n] : st_i = (pk_m, \cdot) \wedge P[i] = \bot$: Stop
Return pk_m

Oracle Respond(j, c, ch)
Require $j \in [m]$
$y \leftarrow_\$ \mathbb{Z}_p$
$h \leftarrow g^y$
If $\exists i \in [n] : P[i] = j \wedge I[i] = c$:
  $(auth, k) \leftarrow \mathrm{H}_1(c, h, pk_j)$
Else:
  $RS[h] \leftarrow y$
  $(auth, k) \leftarrow \mathrm{H}_2(c, h, pk_j)$
$c' \leftarrow (h, auth)$
If $\exists i \in [n] : P[i] = j \wedge I[i] = c$:
  $R[j, i] \xleftarrow{\cup} \{c'\}$
  If $ch \wedge i \notin XP$:
    If $b = 1$: $k \leftarrow_\$ \mathcal{K}$
    $ICH \xleftarrow{\cup} \{i\}$
Return (k, c')

Oracle Corrupt(j)
Require $j \in [m]$
$CR \xleftarrow{\cup} \{j\}$
$sk_j \leftarrow \mathrm{Corrupt}_1(j)$
Return sk_j

Random Oracle $\mathrm{H}(X, Y, pk, Z_1, Z_2)$
If $H[X, Y, pk, Z_1, Z_2] \neq \bot$:
  Return $H[X, Y, pk, Z_1, Z_2]$
If $\exists j \in [m] : pk = pk_j \wedge \mathrm{DDH}(X, pk_j, Z_2)$:
  If $\exists i \in [n] : I[i] = X$:
    $(\cdot, x) \leftarrow st_i$
    If $\mathrm{DDH}(X, Y, Z_1)$:
      If $Y \in R[j, i]$: Return $\mathrm{H}_1(X, Y, pk)$
      Return $\mathrm{H}_3(X, Y, pk)$
  Else if $RS[Y] \neq \bot \wedge Z_1 = X^{RS[Y]}$:
    Return $\mathrm{H}_2(X, Y, pk)$
$(auth, k) \leftarrow_\$ \mathcal{K}^2$
$H[X, Y, pk, Z_1, Z_2] \leftarrow (auth, k)$
Return $(auth, k)$

Fig. 13. Adversary $\mathcal{B}_{1,b}$ against $(q_{\mathrm{gen}}, q_{\mathrm{init}})$-St-CDH.

Adversary $\mathcal{B}_2$ is given in Figure 14 and proceeds similar to $\mathcal{B}_2$ from the proof of Theorem 1.

Similar to the last step in the proof of Theorem 1, we observe that the only place where the two games differ is the challenge in oracle Respond and that as long as $\mathcal{A}$ does not query the random oracle on the respective input, the two games proceed identically. We give a complete description of adversary $\mathcal{B}_2$ in Figure 14. It proceeds similar to $\mathcal{B}_2$ from Theorem 1.

To simulate Init, it queries Gen_1 and answers queries to Expose using oracle $\mathrm{Corrupt}_1$. When $\mathcal{A}$ queries Respond and there exists a partnered initiator instance, then $\mathcal{B}_2$ queries its Gen_2 oracle and uses H_1 to compute the authentication tag and the session key. If $\mathcal{A}$ queries H on (X, Y, pk, Z_1, Z_2) such that X was output by an initiator instance that was not exposed, Y was output by a partnered responder instance and Z_1 is the correct Diffie-Hellman solution, then the call to DDH will set the win flag in $\mathcal{B}_2$'s game. Note also that $\mathcal{B}_2$ does not require access to $\mathrm{Corrupt}_2$.

This concludes the description of $\mathcal{B}_2$. The theorem now follows from collecting the probabilities and folding adversaries $\mathcal{B}_{1,0}$ and $\mathcal{B}_{1,1}$ into a single adversary $\mathcal{B}_1$. $\square$

Adversary $\mathcal{B}_2^{\mathrm{Gen}_1,\mathrm{Gen}_2,\mathrm{Corrupt}_1,\mathrm{DDH}}$
$(n, m) \leftarrow 0^2$
$(P[\cdot], I[\cdot], H[\cdot], H'[\cdot]) \leftarrow \perp^4$
$(Q, \mathit{ICH}, \mathit{RCH}, \mathit{XP}, \mathit{CR}, R[\cdot]) \leftarrow \emptyset^6$
$b' \leftarrow_\$ \mathcal{A}$
Stop with b'

Oracle $\mathrm{Init}(pk)$
$n \leftarrow n + 1$
$x \leftarrow_\$ \mathbb{Z}_p$
$c \leftarrow \mathrm{Gen}_1$
$st_n \leftarrow (pk, \perp)$
If $\exists j \in [m] : pk = pk_j$:
$P[n] \leftarrow j$; $I[n] \leftarrow c$
Return c

Oracle $\mathrm{Receive}(i, c, ch)$
Require $i \in [n] \setminus Q$
$Q \xleftarrow{\cup} \{i\}$
If $c \in R[P[i], i]$: Return
$(pk, x) \leftarrow st_i$
$(h, \mathit{auth}) \leftarrow c$
If $H'[g^x, h, pk] = \perp$: Return $\perp$
$(\mathit{auth}', k) \leftarrow \mathrm{H}_3(g^x, h, pk)$
If $\mathit{auth}' \neq \mathit{auth}$:
$k \leftarrow \perp$
If $ch \wedge i \notin \mathit{XP} \wedge P[i] \notin \mathit{CR} \wedge k \neq \perp$:
Stop
Return k

Oracle $\mathrm{Expose}(i)$
Require $i \in [n] \setminus \mathit{ICH}$
$\mathit{XP} \xleftarrow{\cup} \{i\}$
$(pk, \cdot) \leftarrow st_i$
$x \leftarrow \mathrm{Corrupt}_1(i)$
Return (pk, x)

Internal Random Oracle $\mathrm{H}_i(X, Y, pk)$
If $H_i[X, Y, pk] \neq \perp$: Return $H_i[X, Y, pk]$
$(\mathit{auth}, k) \leftarrow_\$ \mathcal{K}^2$
$H_i[X, Y, pk] \leftarrow (\mathit{auth}, k)$
Return (auth, k)

Oracle Gen
$m \leftarrow m + 1$
$sk_m \leftarrow_\$ \mathbb{Z}_p$
$pk_m \leftarrow g^{sk_m}$
If $\exists i \in [n] : st_i = (pk_m, \cdot) \wedge P[i] = \perp$: Stop
Return pk_m

Oracle $\mathrm{Respond}(j, c, ch)$
Require $j \in [m]$
If $\exists i \in [n] : P[i] = j \wedge I[i] = c$:
$h \leftarrow \mathrm{Gen}_2$
$(\mathit{auth}, k) \leftarrow \mathrm{H}_1(c, h, pk_j)$
Else:
$y \leftarrow_\$ \mathbb{Z}_p$
$h \leftarrow g^y$
$RS[h] \leftarrow y$
$(\mathit{auth}, k) \leftarrow \mathrm{H}_2(c, h, pk_j)$
$c' \leftarrow (h, \mathit{auth})$
If $\exists i \in [n] : P[i] = j \wedge I[i] = c$:
$R[j, i] \xleftarrow{\cup} \{c'\}$
If $ch \wedge i \notin \mathit{XP}$:
If $b = 1$: $k \leftarrow_\$ \mathcal{K}$
$\mathit{ICH} \xleftarrow{\cup} \{i\}$
Return (k, c')

Oracle $\mathrm{Corrupt}(j)$
Require $j \in [m]$
$\mathit{CR} \xleftarrow{\cup} \{j\}$
Return sk_j

Random Oracle $\mathrm{H}(X, Y, pk, Z_1, Z_2)$
If $H[X, Y, pk, Z_1, Z_2] \neq \perp$:
Return $H[X, Y, pk, Z_1, Z_2]$
If $\exists j \in [m] : pk = pk_j \wedge Z_2 = X^{sk_j}$:
If $\exists i \in [n] : I[i] = X$:
$(\cdot, x) \leftarrow st_i$
If $\mathrm{DDH}(X, Y, Z_1)$:
If $Y \in R[j, i]$: Return $\mathrm{H}_1(X, Y, pk)$
Return $\mathrm{H}_3(X, Y, pk)$
Else if $RS[Y] \neq \perp \wedge Z_1 = X^{RS[Y]}$:
Return $\mathrm{H}_2(X, Y, pk)$
$(\mathit{auth}, k) \leftarrow_\$ \mathcal{K}^2$
$H[X, Y, pk, Z_1, Z_2] \leftarrow (\mathit{auth}, k)$
Return (auth, k)

Fig. 14. Adversary $\mathcal{B}_2$ against $(q_{\mathrm{init}}, q_{\mathrm{resp}})$-St-CDH.