Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Mathematical Sciences

Amund Eggen Svandal

Bidirectional Ratcheted Key Exchange Using Broadcasting

Master's thesis in Applied Physics and Mathematics
Supervisor: Jiaxin Pan
June 2023

NTNU
Norwegian University of
Science and Technology

Amund Eggen Svandal

Bidirectional Ratcheted Key Exchange Using Broadcasting

Master's thesis in Applied Physics and Mathematics
Supervisor: Jiaxin Pan
June 2023

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Mathematical Sciences

Bidirectional Ratcheted Key Exchange Using Broadcasting

Amund Eggen Svandal[1]

Department of Mathematical Sciences
postmottak@math.ntnu.no

Abstract. We expand the definition of ratcheted key exchange given by Bellare, Singh, Jaeger, Nyayapati, and Stepanovs to allow bidirectional communication using broadcasting. We give a concrete scheme for ratcheted key exchange using broadcasting and prove that it satisfies our new security definition in the random oracle model.

Keywords. Bidirectional ratcheted key exchange, Oracle Diffie-Hellman, Random oracle model

Table of Contents

1 Introduction

Traditionally, the study of secure messaging in cryptography has concerned itself with the communication of two *trusted* parties, often endearingly called Alice and Bob, over an *insecure* line. The adversary, Eve, has the ability to either eavesdrop, thereby the name, or possibly alter the communication on this line. Though this threat model can give strong guarantees about the information leaked to third parties, it makes no claims about the outcome in case one of the legitimate participants are compromised. This is particularly relevant due to the prevalence of malware and system vulnerabilities like Heartbleed and Logjam [11] [4], and the present trend of mobile computing. A scheme that is secure against third party eavesdroppers could totally break down if the encryption keys stored on a trusted server were leaked, or if the mobile phone used for the communications was lost. This situation motivates an even stronger threat model; where even the participating parties are not trusted, and secrets stored by them may be leaked.

1.1 Ratcheting and Ratcheted Key Exchange

A recent technique that has been developed to mitigate issues related to exposure of secrets is that of *ratcheting.* The basic idea is that participating parties don't store long term secrets (like encryption keys) that could be used to compromise the security of an entire conversation, but rather use a *ratcheting mechanism* to frequently update these secrets. Then in the event of an exposure, only messages encrypted with the current key can be decrypted, and old messages encrypted with old keys cannot. Essential to this approach is that the ratcheting mechanism is *one way*, i.e. it is difficult to recover previous keys even knowing the current one, thereby the name.

An early utilization of ratcheting as a technique was the OTR (Off the Record) communication protocol by Borisov, Goldberg, and Brewer in [9]. The technique has since matured, and is used e.g. in the Signal protocol [16] by Open Whisper Systems which in turn is used by many popular instant messaging services, like WhatsApp [15], to enable end to end encrypted messaging. In [8], Bellare et. al. lift ratcheting from a technique to the cryptographic primitive *ratcheted key exchange*, and formalize syntax, correctness, and security definitions and give a provably secure scheme. The authors restrict themselves to considering single, one-sided ratcheted key exchange, which is a simplification of protocols like Signal, which are two-sided and doubly ratcheted.

1.2 Our Contributions

In this paper we lift one of the restrictions from [8] by creating a provably secure *two-sided* singly ratcheted scheme. We achieve this by using broadcasting; doing the computation in two rounds, instead of one. We define syntax, correctness and security definitions for ratcheted key exchange schemes using broadcasting, and give a provably secure scheme using these definitions. The security proof reduces the security of the scheme to a novel computational assumption IODHE. This assumption is based on ODHE from [8], and we give a reduction from IODHE to CDH in a symmetric (type 1) pairing group.

1.3 Selected Related Works

In [17], Poettering and Rösler propose a provably secure, bidirectional ratcheted key exchange scheme using a HIBE-like component. Their scheme uses single round communication, where *one* message over the wire results in a new key, and so their communication protocol differs from the one we will develop. The authors also seem to manage stronger security guarantees than what we will develop using broadcasting, as they allow the adversary to query exposures for old keys. In relation to this, the authors state that the scheme from [8] does not satisfy their related security definition for unidirectional ratcheted key exchange. In [10], Betul Dürak and Vaudenay propose a bidirectional ratcheted key exchange scheme that is more efficient than that proposed in [17] at the expense of having what they call "slightly sub-optimal security". In [6], Balli, Rösler and Vaudenay discuss the necessity of strong primitives, like those in [17], to construct ratcheted key exchange schemes with strong security guarantees, and give an even stronger security definition for unidirectional schemes involving the manipulation of randomness by the adversary.

2 Preliminaries

2.1 Syntax

We use the framework of code based games from [7]. For a game G and an adversary $\mathcal{A}$ we let $\mathrm{G}^{\mathcal{A}} \Rightarrow 1$ be the event that game G outputs 1 when run by adversary $\mathcal{A}$. As a shorthand, we define $\Pr\left[\mathrm{G}^{\mathcal{A}}\right] = \Pr\left[\mathrm{G}^{\mathcal{A}} \Rightarrow 1\right]$. In figures defining multiple games, lines specific to certain games are annotated with a box naming those games. For two strings $s, t \in \{0,1\}^*$ we let $s \| t$ be the concatenation of the two strings. The concatenation operator is defined to produce uniquely decodable strings to prevent trivial string padding attacks. For $s \in \{0,1\}^*$ we define $s[i]$ to be the i-th bit of s and $s[i \dots j]$ to be the concatenation of bits i through j inclusive. When T is a table we define $T[i]$ to be the element in T indexed by i. We use $\perp$ to denote an empty table position, as well as a special error value that may be returned by algorithms to indicate an error condition. We use $\leftarrow$ as the assignment operator and $\overset{?}{=}$, $\overset{?}{\neq}$ as comparison operators. For finite sets S we let $v \overset{\$}{\leftarrow} S$ denote assigning a uniformly random value from S to v. When running an adversary or an algorithm F, we use $v \overset{\$}{\leftarrow} \mathrm{F}$ when assigning to v the output of F to denote that F may be non-deterministic.

2.2 Pairing Groups

We concern ourselves with symmetric (type 1), cyclic, pairing groups $\mathcal{PG}$. These are described by a tuple $(\mathbb{G}, \mathbb{G}_t, g, p, e)$, where $\mathbb{G}$ is the source group, $\mathbb{G}_t$ is the target group, g is a generator of $\mathbb{G}$, p is the order of $\mathbb{G}$ and $e : \mathbb{G}^2 \rightarrow \mathbb{G}_t$ is a cryptographic bilinear mapping. Viewing both $\mathbb{G}$ and $\mathbb{G}_t$ as multiplicative groups and denoting the unity in $\mathbb{G}_t$ as $1_{\mathbb{G}_t}$, e must satisfy the following:

1. Bilinearity: $e\left(g^x, g^y\right) = e\left(g, g\right)^{xy} \,\forall x, y \in \mathbb{Z}_p$
2. Non-denegeracy: $e\left(g, g\right) \neq 1_{\mathbb{G}_t}$
3. Computability: $e\left(A, B\right)$ must be efficiently computable for all $A, B \in \mathbb{G}$

In algorithms intended to work for generic pairing groups we use the syntax $\mathcal{PG} \overset{\$}{\leftarrow} \mathrm{PGGen}\left(1^\lambda\right)$ to denote the choice of a random pairing group with group size corresponding to the security parameter λ.

2.3 MAC Schemes and Strong Unforgeability under Chosen Message Attack

A MAC scheme F defines a key length F.kl and an evaluation function F.Ev. F.Ev takes a key $fk \in \{0,1\}^{\mathrm{F.kl}}$ and a message from some message space $\mathcal{M}$, and returns a tag σ from some tag space $\mathcal{T}$. In this paper we only consider schemes where F.Ev is deterministic. A MAC scheme F is said to have SUFCMA security if every adversary against the SUFCMA game for F using resources bounded by poly (λ) has an advantage that is negligible in λ, the security parameter. We define the advantage of adversary $\mathcal{F}$ against the SUFCMA game for F to be $\mathrm{Adv}^{\mathrm{SUFCMA}}_{\mathrm{F},\mathcal{F}} = \Pr\left[\mathrm{SUFCMA}^{\mathcal{F}}_{\mathrm{F}}\right]$. We use the definition of the SUFCMA game given in [8]. The game is shown in Figure 1.

```
Game $\mathrm{SUFCMA}^{\mathcal{F}}_{\mathrm{F}}$
$fk \xleftarrow{\$} \{0,1\}^{\mathrm{F.kl}}$
$\mathrm{win} \leftarrow \mathrm{false}$
$\mathcal{F}^{\textsc{Tag},\textsc{Verify}}$
return win

Oracle Tag $(m)$
$\sigma \leftarrow \mathrm{F.Ev}(fk, m)$
$S \leftarrow S \cup \{(m, \sigma)\}$
return $\sigma$

Oracle Verify $(m, \sigma)$
$\sigma^* \leftarrow \mathrm{F.Ev}(fk, m)$
if $\sigma \overset{?}{=} \sigma^*$ and $(m, \sigma) \notin S$ then
    $\mathrm{win} \leftarrow \mathrm{true}$
return $\sigma \overset{?}{=} \sigma^*$
```

Fig. 1. The game defining the strong unforgeability under chosen message attack assumption for MAC scheme F. To win, the adversary $\mathcal{F}$ must compute a valid tag σ on a message m not used in a call to Tag.

2.4 Computational Diffie-Hellman in a Symmetric Pairing Group

The standard computational Diffie-Hellman (CDH) assumption for a group $\mathbb{G}$ states, roughly, that given g, g^x, g^y it is difficult to compute g^{xy}. Extending this assumption to a symmetric pairing group we get the bilinear Diffie-Hellman problem (BDH), stating that given $g, g^x, g^y, g^z \in \mathbb{G}$ it is difficult to compute $e(g, g)^{xyz}$. In this paper we will use the regular CDH assumption in a symmetric pairing group. The game for this assumption is given in Figure 2. It is clear that CDH in a symmetric pairing group is a weaker assumption, implied by BDH. In particular, given an adversary $\mathcal{A}$ that solves CDH in $\mathbb{G}$, we can make a trivial reduction that solves BDH by computing $e(\mathcal{A}(g^x, g^y), g^z)$.

```
Game $\mathrm{CDH}^{\mathcal{D}}$
$\mathcal{PG} \xleftarrow{\$} \mathrm{PGGen}(1^\lambda)$
$(\mathbb{G}, \mathbb{G}_t, g, p, e) \leftarrow \mathcal{PG}$
$x \xleftarrow{\$} \mathbb{Z}_p$
$y \xleftarrow{\$} \mathbb{Z}_p$
$Z \xleftarrow{\$} \mathcal{D}(g, g^x, g^y, \mathcal{PG})$
return $Z \overset{?}{=} g^{xy}$
```

Fig. 2. The game defining the Computational Diffie-Hellman assumption in a type 1 pairing group. To win, the adversary $\mathcal{D}$ must compute g^{xy} in $\mathbb{G}$.

3 Independent Oracle Diffie-Hellman with Exposures

We define the Independent Oracle Diffie-Hellman with Exposures (IODHE) assumption based on the ODHE assumption from [8]. The ODHE assumption states that it is difficult for an adversary to distinguish hashes of g^{xy} from random strings, somewhat similarly to the hashed Diffie-Hellman assumption described in [2]. In ODHE, however, the hash function is given additional input, and the adversary can request for multiple values of $g^{x[i]}$ where it can either try to distinguish the hash of $g^{x[i]y}$ or query an exposure to obtain $x[i]$. We extend this assumption in IODHE by removing the static secret y, and returning a pair $g^{x[i]}, g^{y[i]}$ to the adversary at each new index. When querying an exposure the adversary gets both $x[i]$ and $y[i]$, and some extra bookkeeping is required to prevent trivial attacks. The code for IODHE is given in Figure 3, and we say that a group $\mathbb{G}$ and a hash function family H has IODHE security if the advantage $\mathrm{Adv}^{\mathrm{IODHE}}_{\mathbb{G},\mathrm{H},\mathcal{A}} = 2\Pr\left[\mathrm{IODHE}^{\mathcal{A}}_{\mathbb{G},\mathrm{H}} \Rightarrow 1\right] - 1$ is negligible in the security parameter λ.

Like the ODHE game, IODHE initializes the challenge bit b, a hash key hk, and a generator g to uniformly random values, and the current index v to -1. The Up oracle increments the index v, clears the current operation `op`, generates and stores two random integers $x[i], y[i] \in \mathbb{Z}_p$, and returns $g^{x[i]}, g^{y[i]}$. The Exp oracle checks that the current operation `op` is not "ch" and, if this holds, sets the current operation to "exp" and returns the secrets at the current index $x[v]$ and $y[v]$. Oracles Ch and Hash perform the same duties as in ODHE, but require some additional machinery to present additional capabilities to the adversary while preventing trivial attacks. The Hash oracle takes an index i, a group element X and an exponent that is restricted to the values "x" and "y". With these values it attempts to compute $\mathrm{H.Ev}\left(hk, i \parallel X^{x[v]}\right)$ or $\mathrm{H.Ev}\left(hk, i \parallel X^{y[v]}\right)$ depending on the value of the parameter `exponent`, as long as it would not enable a trivial attack. The Ch oracle returns either $\mathrm{H.Ev}\left(hk, v \parallel g^{x[v]y[v]}\right)$ or a random string depending on the value of the challenge bit b, as long as it would not enable a trivial attack. The random strings are stored in a table `mem` so as to be consistent across calls to Ch. The adversary wins the game if it is able to correctly guess the value of the challenge bit b.

The `op` variable is used to prevent trivial attacks like the adversary querying Exp to learn the values of $x[v]$ and $y[v]$ and then querying Ch with full knowledge of what the oracle should return if $b = 1$. Additionally, the game keeps track of the calls made to oracles Hash and Ch to prevent trivial attacks like the adversary querying Ch and then calling $\textsc{Hash}\left(v, g^{x[v]}, \text{"y"}\right)$, which will return the same result when $b = 1$, but different results with high probability when $b = 0$. Oracle Ch keeps track of the value of v for every call made to it, and rejects a call (by returning $\perp$) when the adversary has made a call of the form $\left(v, g^{x[v]}, \text{"y"}\right)$ or $\left(v, g^{y[v]}, \text{"x"}\right)$ to Hash. Oracle Hash similarly keeps track of the values of i, X, and `exponent` for every call made to it, and rejects a call (by returning $\perp$) when the adversary has made a call on index i to Ch such that $X = g^{x[i]} \wedge \text{exponent} = \text{"y"}$ or $X = g^{y[i]} \wedge \text{exponent} = \text{"x"}$.

We do not know of a pair $\mathbb{G}, \mathrm{H}$ for which this assumption holds in the standard model, but we give a reduction in Appendix A to an intermediary assumption ISCDHE in the random oracle model, followed by a standard model reduction to another intermediary assumption ISCDH and finally to the CDH assumption for a type 1 pairing group.

```
Game $\text{IODHE}^{\mathcal{A}}_{\mathbb{G},\mathrm{H}}$
$b \xleftarrow{\$} \{0,1\}$
$hk \xleftarrow{\$} \{0,1\}^{\mathrm{H.kl}}$
$g \xleftarrow{\$} \mathbb{G}^*$
$v \leftarrow -1$
$b' \xleftarrow{\$} \mathcal{A}^{\textsc{Up},\textsc{Ch},\textsc{Exp},\textsc{Hash}}(hk, g)$
return $b' \overset{?}{=} b$

Oracle Up
$\mathrm{op} \leftarrow \perp$
$v \leftarrow v + 1$
$x[v] \xleftarrow{\$} \mathbb{Z}_p$
$y[v] \xleftarrow{\$} \mathbb{Z}_p$
return $\left(g^{x[v]}, g^{y[v]}\right)$

Oracle Exp
if $\mathrm{op} \overset{?}{=}$ "ch" then
    return $\perp$
$\mathrm{op} \leftarrow$ "exp"
return $(x[v], y[v])$

Oracle Ch
if $\left(\mathrm{op} \overset{?}{=} \text{"exp"}\right)$ or
    $\left(\left(v, g^{x[v]}, \text{"y"}\right) \in S_{\mathrm{hash}}\right)$ or
    $\left(\left(v, g^{y[v]}, \text{"x"}\right) \in S_{\mathrm{hash}}\right)$ then
    return $\perp$
$\mathrm{op} \leftarrow$ "ch"
$S_{\mathrm{ch}} \leftarrow S_{\mathrm{ch}} \cup \{v\}$
$e \leftarrow g^{x[v]y[v]}$
if $\mathrm{mem}[v] \overset{?}{=} \perp$ then
    $\mathrm{mem}[v] \xleftarrow{\$} \{0,1\}^{\mathrm{H.ol}}$
$r_0 \leftarrow \mathrm{mem}[v]$
$r_1 \leftarrow \mathrm{H.Ev}(hk, v \parallel e)$
return $r_b$

Oracle Hash$(i, X, \mathrm{exponent})$
if exponent $\overset{?}{=}$ "x" then
    if $i \in S_{\mathrm{ch}}$ and $X \overset{?}{=} g^{y[i]}$ then
        return $\perp$
    $e \leftarrow X^{x[v]}$
else if exponent $\overset{?}{=}$ "y" then
    if $i \in S_{\mathrm{ch}}$ and $X \overset{?}{=} g^{x[i]}$ then
        return $\perp$
    $e \leftarrow X^{y[v]}$
if $i \overset{?}{=} v$ then
    $S_{\mathrm{hash}} \leftarrow S_{\mathrm{hash}} \cup \{(i, X, \mathrm{exponent})\}$
return $\mathrm{H.Ev}(hk, i \parallel e)$
```

Fig. 3. The game defining the Independent Oracle Diffie-Hellman with Exposures (IODHE) assumption. The game is parametrized by a group $\mathbb{G}$ and a keyed hash function H with key length H.kl, output length H.ol, and evaluation function H.Ev. To win, the adversary $\mathcal{A}$ must distinguish the real ($b = 1$) and random ($b = 0$) operation of the game.

4 Ratcheted Key Exchange Using Broadcasting

Ratcheted key exchange allows users to negotiate and re-negotiate shared secrets. Our work is based on [8], where the authors define correctness and security for two party single ratcheted key exchange schemes, and give a scheme that is provably secure in their security model. Their definition designates one party as the *sender* and the other as the *receiver*, and restricts communication to be strictly one way. In our work we lift this restriction by using broadcasting; each party shares a value with the other that can then be used to compute the new shared secret, much in the spirit of ordinary Diffie-Hellman key exchange. Note that we will only consider schemes with one broadcasting step, but the definitions can be extended to handle multiple steps, potentially allowing for stronger security or more than 2 parties.

4.1 Definition of Ratcheted Key Exchange Using Broadcasting

A ratcheted key exchange scheme using broadcasting BRKE has 3 algorithms: BRKE.IKg, BRKE.BCast, and BRKE.Ratchet. A scheme also defines the key length in bits BRKE.kl, and a randomness space BRKE.RS from which the random seed provided to BRKE.BCast, r, is sampled. BRKE.IKg does the initial key generation and distributes the initial state to the two parties. Each party has a static key stk_i, which is assumed to be public, a session key sek_i, which is private, and the current shared secret k_i, which is also private. We use the subscript i to denote to which party the state belongs, and restrict i to the values 1 and 2 since we are only considering the 2 party case.

Once the two parties have received their initial state, they can start a conversation to generate a new shared secret k. To do this, both parties use BRKE.BCast to generate update information *upd* to be sent to the other party. Once each party receives the update information from the other party they use BRKE.Ratchet to compute the new shared secret k using their own state and the update information received from the other party. Once both parties are done they can use BRKE.BCast and BRKE.Ratchet again to generate the next key.

Syntactically, BRKE.IKg takes no input and returns static- and session keys for both parties, including the initial shared secret: $(k, (stk_1, sek_1, stk_2, sek_2))$. BRKE.BCast takes a party's static- and session keys as well as a random seed as input $(stk, sek; r)$ and returns an updated session key and update information for the other party (sek^*, upd). BRKE.Ratchet takes a party's static- and session keys along with the old shared secret and the update information from the other party as input (stk, sek, k, upd) and returns an updated session key and shared secret along with a flag indicating whether the update information was accepted (sek^*, k, acc).

4.2 Correctness of Ratcheted Key Exchange Schemes Using Broadcasting

When defining correctness for ratcheted key exchange schemes using broadcasting we adapt the correctness definition given in [8] to fit our new setting, while providing essentially the same guarantees. We define correctness using the game BRKE-COR given in Figure 4, and require that for a scheme to be correct, no adversary exists that can win the game.

In BRKE-COR, the adversary $\mathcal{C}$ is given access to two oracles to simulate normal operation of the scheme. Oracle Up accepts the randomness r_1, r_2 used in the oracle and uses BRKE.BCast and BRKE.Ratchet twice to update the state variables of both parties, essentially performing a legitimate key exchange between them. As long as neither party has been tampered with, as is the case in BRKE-COR, we require that the legitimate update information must be accepted and that the keys resulting from the exchange must match. Oracle RatchetO accepts a party $i \in \{1, 2\}$ to perform the ratcheting, a random seed r that is passed to BRKE.BCast, and the update information *upd* that is passed to BRKE.Ratchet. This emulates a party initiating a key exchange and receiving untrusted update information over the internet. We require that a party's session key and shared secret are not changed when update information is rejected by the ratcheting algorithm. Rejecting illegitimate update information is not a requirement for correctness, but is discussed in detail in Section 4.3 on the security of ratcheted key exchange schemes using broadcasting. Note that RatchetO does not alter the state used in Up, and as such cannot be used to tamper with the protocol.

Implementations of ratcheted key exchange using broadcasting should be careful to only ratchet when they have first made a broadcast, and to not broadcast multiple update informations at the same time, but this is not included in the correctness definition.

4.3 Security of Ratcheted Key Exchange Schemes Using Broadcasting

For the security of ratcheted key exchange schemes using broadcasting we desire certain strong security guarantees even in the face of adversaries with complete control of the

```
Game BRKE-COR^C
bad ← false
(k, (stk_1, sek_1, stk_2, sek_2)) ←$ BRKE.IKg
k_1, k_2 ← k, k
C^{Up,RatchetO}(stk_1, stk_2)
return bad =? false

Oracle Up(r_1, r_2)
(sek_1, upd_1) ← BRKE.BCast(stk_1, sek_1; r_1)
(sek_2, upd_2) ← BRKE.BCast(stk_2, sek_2; r_2)
(k_1, sek_1, acc_1) ← BRKE.Ratchet(stk_1, sek_1, upd_1)
(k_2, sek_2, acc_2) ← BRKE.Ratchet(stk_2, sek_2, upd_2)
if not acc_1 or not acc_2 or k_1 ≠? k_2 then
    bad ← true

Oracle RatchetO(i, r, upd)
(sek'_i, upd_1) ← BRKE.BCast(stk_i, sek_i; r)
(k*_i, sek*_i, acc) ← BRKE.Ratchet(stk_i, sek'_i, upd)
if not acc and (sek'_i ≠? sek*_i or k_i ≠? k*_i) then
    bad ← true
```

Fig. 4. Correctness for two party, single broadcast, ratcheted key exchange schemes.

communication channel, and the ability to read the secrets stored by the participants. An adversary with these capabilities should not be able to distinguish generated keys from random strings or make the two parties agree on different keys. Additionally, when the adversary reads the secrets of one party we desire that forward and backward security hold. Like in [8] we define forward security to be the property that even knowing a current key, distinguishing prior keys from random strings is difficult. Similarly for backward security; even knowing a current key it is difficult to distinguish keys generated in the future from random strings.

The security game BKIND is given in Figure 5, and we define that a ratcheted key exchange scheme using broadcasting BRKE has BKIND security if any adversary $\mathcal{D}$ against $\mathrm{BKIND}_{\mathrm{BRKE}}$, making an amount of queries to each oracle that is bounded by a polynomial of the security parameter λ, has advantage $\mathrm{Adv}^{\mathrm{BKIND}}_{\mathrm{BRKE},\mathcal{D}} = 2\Pr\left[\mathrm{BKIND}^{\mathcal{D}}_{\mathrm{BRKE}}\right] - 1$ that is negligible in λ. The security game is adapted from [8], and therefore follows the two basic policies defined there. The first is that whenever update information generated by the adversary is accepted by either party, full knowledge of the key generated by that party should not leak any information about past or future keys. Additionally, when update information generated by one party is accepted by the other, the two parties should agree on the resulting key, and the adversary should not be able to distinguish it from a random string. In adapting the security definition from [8], we also strengthen it by allowing exposure of the secrets of both parties, unlike in the original paper where only exposure of the sender's secrets is permitted.

In game BKIND from Figure 5 we give the adversary access to oracles to run broadcasting and ratcheting for either party, an oracle to read the secrets of either party, and an oracle to read the stored key k_i of either party. Note that the oracles only accept values from $\{1, 2\}$ for i. Oracle BCastO generates a random seed r and updates the state of party i by running BRKE.BCast. It then stores the generated update information to the table `auth`, and increments the index s_i, which is the index of the key which will be generated with this update information. It then returns the update information to the adversary. A check is made at the start to ensure that BRKE.BCast is not called multiple times at the same index.

Oracle RatchetO updates the state of party i by running BRKE.Ratchet using the update information provided by the adversary. If the update information is accepted, it checks the value of the `op` table at the current index to see if the adversary has exposed the secrets of either party at this index. If the adversary has done this, the flag restricted$_i$ is set to `true`, as it is assumed that the adversary has used this information to submit forged update information, and we therefore don't expect the output key to appear random or match the key of the other party. If the submitted update information matches that sent by the other party at this index, however, the restricted flag is set to `false` due to our second policy. This logic assures that restricted$_i$ is `false` whenever the key computed by party i should look random, and is `true` when the adversary has used the Exp oracle to masquerade as the other party and can itself compute the key. Finally, the index c_i is incremented, which is the index of the current key for party i. A check is made at the start to ensure that BRKE.Ratchet is not called multiple times at the same index.

Oracle Exp returns to the adversary the last random seed used in BCastO, r_i, the session key sek_i and the shared secret k_i for party i. To prevent trivial attacks, the oracle checks that the adversary hasn't made a challenge query on the relevant keys and then sets values in the `op` table to note that the adversary queried an exposure. Note that we check and set the `op` table at both indicies s_i *and* c_i due to the fact that one party can store compromising information about two keys at the same time. If the adversary calls Exp directly after a call to RatchetO, s_i and c_i will be equal, and the behavior is identical to that in [8]. If, however, the adversary first calls BCastO, party i may store a secret that can be used to compute the next key from the incoming update information. When the adversary then calls Exp it will get both the current key, and the information necessary to compute the next key. To prevent a trivial attack we therefore also check and set the `op` table at index s_i, which will be $c_i + 1$ in this case.

Oracle Ch returns either the key stored by party i or a random string, depending on the challenge bit b. Before this, however, it first checks the flag restricted$_i$, and if it is `true` returns the real key regardless of the challenge bit. This is because when the flag is `true`, the adversary may know the true value of the key, and could easily distinguish it from a random string. The oracle then checks the `op` table to see if the current key has been exposed, and sets the value in the table to "ch" if it has not, before it finally returns.

Note that we in oracle BCastO set op $[c_i]$ to "ch" when it is not already set. This is not a design feature of the security game, but is an artificial weakness, implanted to enable the security proof given in Section 4.5. In practice, this means that the adversary can only query exposures for a party before they have generated their next update information, and that the adversary cannot query exposures for a party if the other party has ratcheted to a later key (has a larger value of c_i). The exceptions to this are indicies where the `op` table already contain "ch". In these cases, calls to BCastO will not alter the `op` table, and the adversary may query an exposure after calling BCastO, or query an exposure on a party that is far behind its counterpart.

This restriction disallows adversaries that look at the generated update information before deciding whether to query an exposure or not and adversaries that query exposures on previous indicies where one wasn't already queried. Note that the adversary is still able to expose the session key after a call to BCastO by calling Exp, BCastO and then Exp again. The purpose of the change is described in detail in the security proof. This is a small, but significant, weakening of security; further work could aim to remove this artificial weakness.

```
Game $\mathrm{BKIND}^{\mathcal{D}}_{\mathrm{BRKE}}$
b $\xleftarrow{\$} \{0,1\}$
$(k, (stk_1, sek_1, stk_2, sek_2)) \xleftarrow{\$}$ BRKE.IKg
$k_1, k_2 \leftarrow k, k$
$b' \xleftarrow{\$} \mathcal{D}^{\mathrm{BCastO},\mathrm{RatchetO},\mathrm{Exp},\mathrm{Ch}}(stk_1, stk_2)$
return $b' \overset{?}{=} b$

Oracle BCastO $(i)$
if $s_i \overset{?}{\neq} c_i$ then
    return $\perp$
if $\mathrm{op}[c_i] \overset{?}{=} \perp$
    $\mathrm{op}[c_i] \leftarrow$ "ch"
$r_i \xleftarrow{\$}$ BRKE.RS
$(sek_i, upd) \leftarrow$ BRKE.BCast $(stk_i, sek_i; r_i)$
auth $[i]\,[c_i] \leftarrow upd$
$s_i \leftarrow s_i + 1$
return $upd$

Oracle RatchetO $(i, upd)$
if $s_i \overset{?}{=} c_i$ then
    return false
$(k_i, sek_i, acc) \leftarrow$ BRKE.Ratchet $(stk_i, sek_i, upd)$
if not $acc$ then
    return false
if $\mathrm{op}[c_i] \overset{?}{=}$ "exp"
    $\mathrm{restricted}_i \leftarrow$ true
if $upd \overset{?}{=}$ auth $[3-i]\,[c_i]$ then
    $\mathrm{restricted}_i \leftarrow$ false
$c_i \leftarrow c_i + 1$
return true

Oracle Exp $(i)$
if $\mathrm{op}[c_i] \overset{?}{=}$ "ch" then
    return $\perp$
if $\mathrm{op}[s_i] \overset{?}{=}$ "ch" then
    return $\perp$
$\mathrm{op}[c_i] \leftarrow$ "exp"
$\mathrm{op}[s_i] \leftarrow$ "exp"
return $(r_i, sek_i, k_i)$

Oracle Ch $(i)$
if $\mathrm{restricted}_i$ then
    return $k_i$
if $\mathrm{op}[c_i] \overset{?}{=}$ "exp" then
    return $\perp$
$\mathrm{op}[c_i] \leftarrow$ "ch"
if $\mathrm{rkey}[c_i] \overset{?}{=} \perp$ then
    $\mathrm{rkey}[c_i] \xleftarrow{\$} \{0,1\}^{\mathrm{RKE.kl}}$
if $b \overset{?}{=} 1$ then
    return $k_i$
return $\mathrm{rkey}[c_i]$
```

Fig. 5. The security of key indistinguishability for ratcheted key exchange schemes using broadcasting.

4.4 Ratcheted Key Exchange Scheme Using Broadcasting

Our ratcheted key exchange scheme using broadcasting is based on the scheme from [8], and is adapted to fit the new protocol using broadcasting, and to achieve the new security of BKIND. In [8], the sender generates a secret exponent x, and sends g^x to the receiver, and the key is computed by combining this with the receiver's long term private/public pair y and g^y to get g^{xy} and then hashing this value. This mechanism is similar to the key generation used for the encryption scheme DHIES from [1], which is essentially an implementation of hashed Diffie-Hellman, described in [2]. To avoid certain trivial attacks, described in Section 4.3 of [8], the scheme also includes the current index and the update information sent by the sender to the receiver as additional input to the hash function used to derive the key.

In our bidirectional scheme we remove long term secret y, and require that both parties generate a secret exponent and send the corresponding group element to the other party at each broadcasting step. The new key is then computed in a similar way as in [8], but using two new group elements at each ratcheting step instead of just one. Like in the original scheme, we also include a MAC tag in the update information to ensure the authenticity of the exchange. Our new scheme is given in Figure 6.

Our scheme derives the key in much the same way as the scheme in [8], but does not include the update information as input to the hash function. Since we have two sets of update information for each new key, a naïve implementation including these as input to the hash function could cause the two parties to disagree on the resulting key, as both parties would need to include them in the same order to get the same result. This is easily fixed by assigning an order to the two parties, such that one party's update information is always included first, but this is somewhat cumbersome, and ultimately unnecessary. The main issue of the scheme from [8] mitigated by the inclusion of the update information as input to the hash function is that of key collision. This occurs when the adversary is able to make the two parties agree on a key when the `restricted` flag is set to `true` for the receiver. When this is the case, challenge queries to the sender will return the real or random key depending on the challenge bit b, whereas challenge queries to the receiver will always return the real key, yielding a good attack. In our scheme, both parties need to receive update information in order to compute the next key, and for this reason it is difficult for an adversary to make the two parties agree on a key while setting the `restricted` flag to `true` for only one of them. In order to set the `restricted` flag to `true` for one party, the adversary must create forged update information using the Exp oracle that is accepted by the party. If the adversary submits update information with a different group element than that generated by the other party, the two parties will disagree on the resulting key, and the adversary cannot make them agree again without altering the `restricted` flag for either party. If the adversary submits update information with the same group element, the MAC tag will be the same as well for the same key, and so `restricted` will be set to `false`. These arguments are formalized in the security proof in Section 4.5.

```
BRKE.IKg
$g \leftarrow \mathbb{G}^*$
$k \leftarrow \{0,1\}^{\mathrm{BRKE.kl}}$
$fk \leftarrow \{0,1\}^{\mathrm{F.kl}}$
$hk \leftarrow \{0,1\}^{\mathrm{H.kl}}$
$stk_1, stk_2 \leftarrow (g, hk), (g, hk)$
$sek_1, sek_2 \leftarrow (0, \perp, fk), (0, \perp, fk)$
return $(k, stk_1, sek_1, stk_2, sek_2)$
```

```
BRKE.BCast $(stk, sek; r)$
$(g, hk) \leftarrow stk$
$(c, x, fk) \leftarrow sek$
$X \leftarrow g^r$
$\sigma \leftarrow \mathrm{F.Ev}(fk, X)$
$upd \leftarrow (X, \sigma)$
$sek^* \leftarrow (c, x, fk)$
return $(sek^*, upd)$
```

```
BRKE.Ratchet $(stk, sek, k, upd)$
$(g, hk) \leftarrow stk$
$(c, x, fk) \leftarrow sek$
$(Y, \sigma) \leftarrow upd$
if $\sigma \overset{?}{\neq} \mathrm{F.Ev}(fk, Y)$ then
    return $(sek, k, \mathrm{false})$
$h \leftarrow \mathrm{H.Ev}(hk, c \parallel Y^x)$
$k^* \leftarrow h[1 \ldots \mathrm{BRKE.kl}]$
$fk^* \leftarrow h[\mathrm{BRKE.kl} + 1 \ldots \mathrm{BRKE.kl} + \mathrm{F.kl}]$
$sek^* \leftarrow (c + 1, \perp, fk^*)$
return $(sek^*, k^*, \mathrm{true})$
```

Fig. 6. Ratcheted key exchange scheme using broadcasting BRKE, parametrized by the group $\mathbb{G}$, the hash function family H, and MAC scheme F.

4.5 Security Proof for the Scheme

Theorem 1. *Given an adversary $\mathcal{D}$ against the* BKIND *security of* BRKE$[\mathbb{G}, \mathrm{H}, \mathrm{F}]$ *making* Q_{R} *queries to* RatchetO*,* Q_{E} *queries to* Exp*, and* Q_{C} *queries to* Ch*, there are adversaries* $\mathcal{O}_1$ *and* $\mathcal{O}_2$ *against the* IODHE *security of* $\mathbb{G}$ *and* H *and an adversary* $\mathcal{F}$ *against the* SUFCMA *security of* F *such that*

$$\begin{aligned}\mathrm{Adv}^{\mathrm{BKIND}}_{\mathrm{BRKE},\mathcal{D}} \leq{} & 2 \cdot (Q_{\mathrm{R}} + 1) \cdot \mathrm{Adv}^{\mathrm{SUFCMA}}_{\mathrm{F},\mathcal{F}} + \\ & 2 \cdot Q_{\mathrm{R}} \cdot \mathrm{Adv}^{\mathrm{IODHE}}_{\mathbb{G},\mathrm{H},\mathcal{O}_1} + \\ & 2 \cdot \mathrm{Adv}^{\mathrm{IODHE}}_{\mathbb{G},\mathrm{H},\mathcal{O}_2}.\end{aligned}$$

Adversary $\mathcal{O}_1$ *makes at most* $Q_{\mathrm{R}} + 1$ *queries to its* IODHE Up *oracle, 2 queries to its* IODHE Ch *oracle,* Q_{E} *queries to its* IODHE Exp *oracle, and* $Q_{\mathrm{R}} - 2$ *queries to its* IODHE Hash *oracle. Adversary* $\mathcal{O}_2$ *makes at most* $Q_{\mathrm{R}} + 1$ *queries to its* IODHE Up *oracle,* $Q_{\mathrm{R}} + 1$ *queries to its* IODHE Ch *oracle,* Q_{E} *queries to its* IODHE Exp *oracle, and* $Q_{\mathrm{R}} + Q_{\mathrm{E}}$ *queries to its* IODHE Hash *oracle. Adversary* $\mathcal{F}$ *makes at most* $Q_{\mathrm{R}} + 1$ *queries to its* SUFCMA Tag *oracle, and* Q_{R} *queries to its* SUFCMA Verify *oracle.*

We prove the security of our scheme using the same method as in [8]. The proof is adapted slightly to work with the updated assumption and security definition. Note that the amount of queries to BCastO is bounded by $Q_{\mathrm{R}} + 1$.

To simulate the random case ($b_{\mathrm{BKIND}} = 0$) of the BKIND game using an IODHE instance, we need to replace the key k_i and the MAC key fk_i with random bits, unless the adversary calls the Exp oracle. However, if the adversary attempts to submit forged update information before it decides whether to query Exp or the challenge oracles, we do not know whether to use the real or the random MAC key to verify the provided update information. To circumvent this issue we first use a hybrid argument with the IODHE assumption and the security of the MAC-scheme to bound the probability of an adversary forging valid update information at any step. In subsequent games, all forgeries are assumed to be invalid, and we can delay committing to the real or random keys until necessary. At this point we make a reduction from this adapted game to the IODHE assumption, which concludes the proof.

Proof (Theorem 1). We prove the theorem by using a sequence of games and relating the probabilities of certain events in these games, like the adversary winning or a flag being set, to the advantage of the adversary against our scheme. These games are:

Game $\mathrm{G}_{0,j}$. This is the same as $\mathrm{BKIND}_{\mathrm{BRKE}}$ with the code for the scheme inserted, but where attempted forgeries for the j first keys are assumed to be invalid. $\mathrm{G}_{0,0}$ is identical to $\mathrm{BKIND}_{\mathrm{BRKE}}$. The code for this game is given in Figure 7.

Game $\mathrm{G}^*_{0,j}$. This is the same as $\mathrm{G}_{0,j}$, but where forgery attempts for the $j+1$-th key are also rejected. This means that $\Pr\left[\mathrm{G}^{*\mathcal{D}}_{0,j}\right] = \Pr\left[\mathrm{G}^{\mathcal{D}}_{0,j+1}\right]$. The code for this game is given in Figure 7.

Game I_j. This is the same as $\mathrm{G}^*_{0,j}$, but where the hash value used to compute the $j+1$-th key is replaced with a uniformly random value. The code for this game is given in Figure 7.

Game G_1. This is identical to game G_{0,Q_R+1}, but has been rewritten to simplify the rest of the proof. The code for this game is given in Figure 8.

Game G_2. This is the same as G_1, but where the value used to create k_i and fk_i is sampled randomly, instead of taken as an output from the hash function. If the adversary calls Exp, the keys are changed to use real hash values. The code for this game is given in Figure 8.

We use the sequences of games $G_{0,j}, G^*_{0,j}$ and I_j in Figure 7 in a hybrid argument to prove that the adversary cannot forge valid update information without calling Exp. This hybrid argument is made exactly akin to the argument in the proof of Theorem 4.1 in [8], however, the games and the corresponding reductions have been adapted to the new assumption and security definition. The specific changes will be noted as they come up.

Like in [8], we add a flag $\texttt{unchanged}_\texttt{i}$, which is `true` when the last update information passed to party i is the update information produced by the other party for that index. Novel in this proof is that both parties get such a flag, hence the subscript i. When this flag is `true`, k_i and fk_i should be indistinguishable from random, and the adversary should not be able to forge update information accepted by party i without first calling Exp. We also add the flag `bad`, which is set to `true` if the adversary submits a valid forgery for the $j+1$-th key.

Since $G_{0,j}$ and $G^*_{0,j}$ are identical until bad, the fundamental lemma of game playing from [7] gives us that

$$\Pr\left[G_{0,j}^{\mathcal{D}}\right] - \Pr\left[G_{0,j}^{*\mathcal{D}}\right] \leq \Pr\left[\text{bad}^{G_{0,j}^{*\mathcal{D}}}\right],$$

where $\text{bad}^{G_{0,j}^{*\mathcal{D}}}$ is the event that the flag `bad` is set to `true` in game $G^*_{0,j}$ when run by adversary $\mathcal{D}$. To bound the probability of this event we use the two adversaries $\mathcal{O}_1$ and $\mathcal{F}$ given in Figures 9 and 10.

Adversary $\mathcal{O}_1$ is a reduction to IODHE that works by first randomly picking an index j' and then perfectly simulating the view of $\mathcal{D}$ in either game $G^*_{0,j'}$ or game $I_{j'}$, depending on the value of the challenge bit in the IODHE instance, b_{IODHE}. It does this by using the IODHE Hash oracle to compute legitimate hash values for all indicies except j', where it uses the IODHE Ch oracle that will either return a real hash value or a random string. If $b_{\text{IODHE}} = 1$ the IODHE Ch oracle returns real hash values, and $\mathcal{O}_1$ perfectly simulates the view of $\mathcal{D}$ in $G^*_{0,j}$. If $b_{\text{IODHE}} = 0$ the IODHE Ch oracle returns random strings, and $\mathcal{O}_1$ perfectly simulates the view of $\mathcal{D}$ in I_j. Special care is taken in Exp to return the correct values for x_i and r_i depending on the time it is called.

Note that if the adversary $\mathcal{D}$ queries an exposure at index j' our reduction $\mathcal{O}_1$ no longer perfectly simulates either $G^*_{0,j'}$ or $I_{j'}$ as we cannot query the IODHE Exp oracle because we have already made a call to the IODHE Ch oracle. This does not matter for our calculations, however, as we know that `bad` will not be set because $\mathcal{D}$ querying Exp prevents this from happening. Inspecting the code of $\mathcal{O}_1$ we see that b', the bit it returns, starts at 0 and is set to 1 exactly when the flag `bad` is set to `true`. Thus, for all $j \in \{1, \ldots, Q_R\}$ we have that

$$\Pr\left[\text{bad}^{G_{0,j}^{*\mathcal{D}}}\right] = \Pr\left[b' = 1 \mid b_{\text{IODHE}} = 1, j' = j\right],$$
$$\Pr\left[\text{bad}^{I_j^{\mathcal{D}}}\right] = \Pr\left[b' = 1 \mid b_{\text{IODHE}} = 0, j' = j\right],$$

and from the definition of advantage we get that

$$\text{Adv}^{\text{IODHE}}_{\mathbb{G},\text{H},\mathcal{O}_1} = \Pr\left[b' = 1 \mid b_{\text{IODHE}} = 1\right] - \Pr\left[b' = 1 \mid b_{\text{IODHE}} = 0\right],$$

which when combined gives us that

$$\begin{aligned}\mathrm{Adv}^{\mathrm{IODHE}}_{\mathbb{G},\mathrm{H},\mathcal{O}_1} &= \Pr\left[b' = 1 \mid b_{\mathrm{IODHE}} = 1\right] - \Pr\left[b' = 1 \mid b_{\mathrm{IODHE}} = 0\right] \\ &= \sum_{j=1}^{Q_\mathrm{R}} \Pr\left[j' = j\right] \left(\Pr\left[\mathrm{bad}^{\mathrm{G}^{*\mathcal{D}}_{0,j}}\right] - \Pr\left[\mathrm{bad}^{\mathrm{I}^{\mathcal{D}}_j}\right]\right) \\ &= \sum_{j=0}^{Q_\mathrm{R}} \frac{\Pr\left[\mathrm{bad}^{\mathrm{G}^{*\mathcal{D}}_{0,j}}\right] - \Pr\left[\mathrm{bad}^{\mathrm{I}^{\mathcal{D}}_j}\right]}{Q_\mathrm{R}}.\end{aligned}$$

Note that we could extend the sum to start at 0, as $\mathrm{G}^*_{0,0}$ and I_0 are identical games, and the probability of `bad` being set in them is equal when run by the same adversary.

To complete the hybrid argument and arrive at game G_1, we bound the probability that the flag `bad` is set to `true` when adversary $\mathcal{D}$ is playing game I_j. In this case, the adversary has forged a valid MAC tag on a message when the MAC key, fk_i, was uniformly random. We use this fact to construct the reduction $\mathcal{F}$ to the SUFCMA security of the MAC scheme F. The reduction $\mathcal{F}$ works similarly to $\mathcal{O}_1$ in that it first picks a random j' (this time from 0 to Q_R) and then perfectly simulates the view of adversary $\mathcal{D}$ in game $\mathrm{I}_{j'}$. The index j' is essentially a guess for when the adversary will attempt their first forgery, and the reduction will simulate tagging and verifying using it's own values for fk_i for every index except j', where it will use the oracles provided by the SUFCMA instance instead. Similarly to $\mathcal{O}_1$, our reduction $\mathcal{F}$ will not perfectly simulate $\mathrm{I}_{j'}$ if the adversary $\mathcal{D}$ queries an exposure at index j' as we do not know the key that the underlying SUFCMA instance is using, and as such don't know what to return for fk_i. This does not matter for our calculations, however, as we know that `bad` will not be set because $\mathcal{D}$ querying Exp prevents this from happening. Examining the code of $\mathcal{F}$ in Figure 10 it is clear that the flag `bad` is set to `true` if and only if adversary $\mathcal{D}$ submits update information with a valid, forged MAC tag at index j'. Thus for $j \in \{0, \ldots, Q_\mathrm{R}\}$ we have that $\Pr\left[\mathrm{bad}^{\mathrm{I}^{\mathcal{D}}_j}\right] = \Pr\left[\mathrm{SUFCMA}^{\mathcal{F}}_{\mathrm{F}} \mid j' = j\right]$, which gives us that $\sum_{j=0}^{Q_\mathrm{R}} \Pr\left[\mathrm{bad}^{\mathrm{I}^{\mathcal{D}}_j}\right] = (Q_\mathrm{R}+1)\,\mathrm{Adv}^{\mathrm{SUFCMA}}_{\mathrm{F},\mathcal{F}}$.

Combining our work we now get that

$$\begin{aligned}\Pr\left[\mathrm{BKIND}^{\mathcal{D}}_{\mathrm{BRKE}}\right] &= \Pr\left[\mathrm{G}^{\mathcal{D}}_{0,0}\right] \\ &= \Pr\left[\mathrm{G}^{\mathcal{D}}_{0,Q_\mathrm{R}+1}\right] + \sum_{j=0}^{Q_\mathrm{R}} \left(\Pr\left[\mathrm{G}^{\mathcal{D}}_{0,j}\right] - \Pr\left[\mathrm{G}^{*\mathcal{D}}_{0,j}\right]\right) \\ &\le \Pr\left[\mathrm{G}^{\mathcal{D}}_1\right] + \sum_{j=0}^{Q_\mathrm{R}} \Pr\left[\mathrm{bad}^{\mathrm{G}^{*\mathcal{D}}_{0,j}}\right] \\ &= \Pr\left[\mathrm{G}^{\mathcal{D}}_1\right] + Q_\mathrm{R}\mathrm{Adv}^{\mathrm{IODHE}}_{\mathbb{G},\mathrm{H},\mathcal{O}_1} + \sum_{j=0}^{Q_\mathrm{R}} \Pr\left[\mathrm{bad}^{\mathrm{I}_j{}^{\mathcal{D}}}\right] \\ &= \Pr\left[\mathrm{G}^{\mathcal{D}}_1\right] + Q_\mathrm{R}\mathrm{Adv}^{\mathrm{IODHE}}_{\mathbb{G},\mathrm{H},\mathcal{O}_1} + (Q_\mathrm{R}+1)\,\mathrm{Adv}^{\mathrm{SUFCMA}}_{\mathrm{F},\mathcal{F}},\end{aligned}$$

where we also used that $\mathrm{G}_{0,Q_\mathrm{R}+1}$ is identical to G_1.

We now move on to games G_1 and G_2. This section of the proof still mostly follows the logic used in [8], but the relevant parts of the games and the reduction $\mathcal{O}_2$ have been completely rewritten to fit the new scheme and security definition. Starting with G_1, this is identical to $\mathrm{G}_{0,Q_\mathrm{R}+1}$, but has been rewritten to simplify the proof. The nested `if` statement at the start of RatchetO has been removed as $c_i < Q_\mathrm{R}+1$ is always true, such that all forgery attempts are rejected. The major difference in syntax between G_{0,Q_R+1} and G_1 is that we in G_1 delay setting the values of k_i and fk_i until they are known by the adversary or we are forced to commit to a value (e.g. for tagging update information). We accomplish this by storing the hash values used for the legitimate conversation between the two parties in a table `hash`. When the flag $\texttt{unchanged}_\texttt{i}$ is `true` we use this table to get the value for fk_i in BCastO so we can create and later verify MAC tags, and to get the value for k_i when we need to answer a challenge query. If the adversary queries an exposure and uses this information to submit forged update information to party i, $\texttt{unchanged}_\texttt{i}$ will be set to `false`, and we switch to directly computing k_i and fk_i as long as the flag is set. When the adversary queries Exp, we grab the last update information that party received, stored in recv_i, and use it to compute and store the hash value to the `hash` table and update the value of k_i and fk_i. These changes gives our reduction $\mathcal{O}_2$ the freedom to not commit to a key before it has to, while making sure that it does not alter the keys after committing to them. Finally, progressing to game G_2, we replace the legitimate hash value computed in RatchetO when $\texttt{unchanged}_\texttt{i}$ is `true` with a random string.

Our reduction $\mathcal{O}_2$ in Figure 11 perfectly simulates the view of adversary $\mathcal{D}$ in either G_1 or G_2, depending on the challenge bit in the underlying IODHE instance: b_{IODHE}. It does this by using the IODHE Ch oracle that returns either the correct hash value, used in G_1, or a random string, used in G_2. We must, however, take great care in *when* we use this oracle, as we must use it for every legitimate key produced by either party, while still allowing the adversary to query an exposure, at which point we must reset back to a real hash value using the IODHE Hash oracle.

Like the definition of game G_1, our reduction $\mathcal{O}_2$ will directly compute k_i and fk_i when $\texttt{unchanged}_\texttt{i}$ is `false`. It does this by using the IODHE Hash oracle. It also uses this oracle in ExpSim along with IODHE Exp to compute the legitimate key and return the last random seed r_i and the secret x_i. Like in $\mathcal{O}_1$, care is taken to return correct values for r_i and x_i based on the current state. In oracle ChSim it uses the IODHE Ch oracle to get the real hash value or a random string, to simulate either G_1 or G_2, and otherwise operates exactly like G_1 and G_2. Unlike G_1 and G_2, however, $\mathcal{O}_2$ does not compute the hash value for the *next* key in RatchetO, but rather uses IODHE Ch to compute the hash value for the *current* key in BCastO. If the adversary first calls either ExpSim or ChSim, however, these oracles will compute the value to store in the `hash` table instead, and BCastO will not call IODHE Ch. The reduction cannot call IODHE Ch in RatchetO, as it does not know whether the adversary will query Ch or Exp after ratcheting. Instead, it delays the call until the adversary calls either Ch or BCastO, or prevents it if the adversary calls Exp. After the potential call to Ch in BCastO, the reduction uses IODHE Up to generate a new pair X, Y to use as update information. The values are stored in a table, and the value corresponding to the party is used. It is important that it calls Up *after* the potential call to Ch in order to keep the `op` table in our reduction in sync with the value of `op` in the underlying IODHE instance. It is also important that we do not call Up any later than at the end of BCastO, like at the start of RatchetO, as this would mean that the reduction would not be able to provide values for r_i when answering ExpSim queries because the IODHE game would have moved on to the next pair X, Y.

When our reduction responds to BCastO queries, it must commit to a value for fk_i to use for tagging the update information. To simulate G_2, the reduction must use a random string for fk_i unless the adversary $\mathcal{D}$ queries an exposure. At the time of a call to BCastO, however, our reduction does not know whether the adversary will query Exp or not and so does not know whether to commit to using a real hash value or a random string for fk_i. The reduction also cannot change the value of fk_i after a call to BCastO as the adversary can easily validate the MAC tag returned by BCastO by computing $\mathrm{F.Ev}(\mathit{fk}_i, X)$, and see that the keys have changed. This is where the artificial weakness we implanted in the security definition in Section 4.3 comes in. This condition sets $\mathrm{op}[c_i]$ to "ch" in BCastO if it has no value yet. This prevents the adversary from calling BCastO followed by Exp for one party when the current key is not already known by the adversary due to an earlier call to Exp. In this case $\mathrm{op}[c_i]$ will be set to "ch", and the call to Exp will be rejected. The reduction exploits this by commiting to a value for fk_i in BCastO, as $\mathrm{op}[c_i]$ will not be empty, and the reduction will know whether to use a the IODHE Ch or Hash oracle to compute the key.

After a call to BCastO, $\mathrm{op}[c_i]$ in our reduction may be "exp" if it had that value before the call, or "ch" otherwise. The value of `op` in the IODHE instance will be $\perp$, as BCastO calls Up. At this point, the adversary may do nothing, or may call ChSim or ExpSim if it is allowed. Note that a call to ChSim after BCastO will not call the IODHE Ch oracle as $\mathrm{op}[c_i]$ cannot be empty. This is important, as BCastO calls Up, and so a call to Ch in ChSim would not only alter the IODHE `op` value but also return the hash value used for the *next* key; not the current one. Doing nothing obviously also does not alter the `op` table in the reduction or the `op` value in the IODHE instance. Once the adversary then calls RatchetO, c_i is incremented and the current value in the reduction's `op` table is empty while the `op` value in the IODHE instance is also empty. If the adversary after the call to BCastO instead calls Exp, the IODHE `op` value will be set to "exp", and the *next* value in the reduction's `op` table will be set to "exp" such that after incrementing c_i in RatchetO, both current values will be "exp". Thus, the `op` values are kept in sync, and all the requests made by $\mathcal{O}_2$ to the IODHE oracles are permitted. The above argument is made with respect to the first party to set the values in the `op` table, as once set they cannot change, and it is clear that the second party will not violate this invariant when catching up to the first. Additionally, queries to the oracles are stored in tables so that when the adversary calls Exp or BCastO on the party that is behind the other, no calls will be made to the IODHE oracles.

The reduction $\mathcal{O}_2$ gets adversary $\mathcal{D}$'s guess, b', for the value of b and when $\mathcal{D}$ guesses correctly assumes that the IODHE Ch oracle was returning real output ($b_{\mathrm{IODHE}} = 1$) and returns $b^* = 1$, otherwise it returns $b^* = 0$. Thus we have that

$$\Pr\left[G_1^{\mathcal{D}}\right] = \Pr\left[b^* = 1 \mid b_{\mathrm{IODHE}} = 1\right]$$
$$\Pr\left[G_2^{\mathcal{D}}\right] = \Pr\left[b^* = 1 \mid b_{\mathrm{IODHE}} = 0\right]$$

and using the definition of advantage we get that $\mathrm{Adv}^{\mathrm{IODHE}}_{\mathbb{G},\mathrm{H},\mathcal{O}_2} = \Pr\left[G_1^{\mathcal{D}}\right] - \Pr\left[G_2^{\mathcal{D}}\right]$. Combining this with our last result we get that

$$\begin{aligned}\Pr\left[\mathrm{BKIND}^{\mathcal{D}}_{\mathrm{BRKE}}\right] &\leq \Pr\left[\mathrm{G}_1^{\mathcal{D}}\right] + Q_{\mathrm{R}}\mathrm{Adv}^{\mathrm{IODHE}}_{\mathbb{G},\mathrm{H},\mathcal{O}_1} + (Q_{\mathrm{R}}+1)\,\mathrm{Adv}^{\mathrm{SUFCMA}}_{\mathrm{F},\mathcal{F}} \\ &= \Pr\left[\mathrm{G}_2^{\mathcal{D}}\right] + \mathrm{Adv}^{\mathrm{IODHE}}_{\mathbb{G},\mathrm{H},\mathcal{O}_2} \\ &\qquad + Q_{\mathrm{R}}\mathrm{Adv}^{\mathrm{IODHE}}_{\mathbb{G},\mathrm{H},\mathcal{O}_1} + (Q_{\mathrm{R}}+1)\,\mathrm{Adv}^{\mathrm{SUFCMA}}_{\mathrm{F},\mathcal{F}}.\end{aligned}$$

Additionally, $\Pr\left[\mathrm{G}_2^{\mathcal{D}}\right] = \frac{1}{2}$, as the view of adversary $\mathcal{D}$ is independent of the challenge bit b. The only place b is referenced in G_2 is in a branch at the end of oracle Ch. The contents of rkey $[c_i]$ are random bits, so for the game to depend on the value of b, k_i cannot be random bits. This can only happen if the second branch in RATCHETO was taken, where k_i is computed using H.Ev. This branch can only be taken, however, when `restricted`$_\mathtt{i}$ is `true`, and as such any call to CH will exit early and not reach the last branch. Thus the view of the game is independent of the challenge bit b such that

$$\begin{aligned}\mathrm{Adv}^{\mathrm{BKIND}}_{\mathrm{BRKE},\mathcal{D}} &= 2\Pr\left[\mathrm{BKIND}^{\mathcal{D}}_{\mathrm{BRKE}}\right] - 1\\ &\leq 2\mathrm{Adv}^{\mathrm{IODHE}}_{\mathbb{G},\mathrm{H},\mathcal{O}_2} + 2Q_{\mathrm{R}}\mathrm{Adv}^{\mathrm{IODHE}}_{\mathbb{G},\mathrm{H},\mathcal{O}_1} + 2\left(Q_{\mathrm{R}}+1\right)\mathrm{Adv}^{\mathrm{SUFCMA}}_{\mathrm{F},\mathcal{F}}.\end{aligned}$$

The bounds on the amount of queries each reduction makes can be found by examining their code. □

```
Games $\mathrm{G}_{0,j}, \mathrm{G}^*_{0,j}, \mathrm{I}_j$
01 $\mathrm{b} \xleftarrow{\$} \{0,1\}$
02 $\mathrm{unchanged}_1 \leftarrow \mathrm{true}$
03 $\mathrm{unchanged}_2 \leftarrow \mathrm{true}$
04 $rand \xleftarrow{\$} \{0,1\}^{\mathrm{H.ol}}$
05 $g \xleftarrow{\$} \mathbb{G}^*$
06 $k \xleftarrow{\$} \{0,1\}^{\mathrm{BRKE.kl}}$
07 $hk \xleftarrow{\$} \{0,1\}^{\mathrm{H.kl}}$
08 $fk \xleftarrow{\$} \{0,1\}^{\mathrm{F.kl}}$
09 $k_1, k_2 \leftarrow k, k$
10 $fk_1, fk_2 \leftarrow fk, fk$
11 $stk_1 \leftarrow (g, hk)$
12 $stk_2 \leftarrow (g, hk)$
13 $s_1, s_2 \leftarrow 0, 0$
14 $c_1, c_2 \leftarrow 0, 0$
15 $x_1, x_2 \leftarrow \bot, \bot$
16 $b' \leftarrow \mathcal{D}^{\textsc{BCastO},\textsc{RatchetO},\textsc{Exp},\textsc{Ch}}(stk_1, stk_2)$
17 return $b' \overset{?}{=} b$

Oracle $\textsc{Exp}(i)$
18 if $\mathrm{op}[c_i] \overset{?}{=}$ “ch” then
19     return $\bot$
20 if $\mathrm{op}[s_i] \overset{?}{=}$ “ch” then
21     return $\bot$
22 $\mathrm{op}[c_i] \leftarrow$ “exp”
23 $\mathrm{op}[s_i] \leftarrow$ “exp”
24 return $(r_i, (c_i, x_i, fk_i), k_i)$

Oracle $\textsc{Ch}(i)$
25 if $\mathrm{restricted}_i$ then
26     return $k_i$
27 if $\mathrm{op}[c_i] \overset{?}{=}$ “exp” then
28     return $\bot$
29 $\mathrm{op}[c_i] \leftarrow$ “ch”
30 if $\mathrm{rkey}[c_i] \overset{?}{=} \bot$ then
31     $\mathrm{rkey}[c_i] \xleftarrow{\$} \{0,1\}^{\mathrm{RKE.kl}}$
32 if $b \overset{?}{=} 1$ then
33     return $k_i$
34 return $\mathrm{rkey}[c_i]$

Oracle $\textsc{BCastO}(i)$
35 if $s_i \overset{?}{\neq} c_i$ then
36     return $\bot$
37 if $\mathrm{op}[c_i] \overset{?}{=} \bot$
38     $\mathrm{op}[c_i] \leftarrow$ “ch”
39 $x_i \xleftarrow{\$} \mathbb{Z}_p$
40 $X \leftarrow g^{x_i}$
41 $\sigma \leftarrow \mathrm{F.Ev}(fk_i, X)$
42 $upd \leftarrow (X, \sigma)$
43 $\mathrm{auth}[i][c_i] \leftarrow upd$
44 $s_i \leftarrow s_i + 1$
45 return $upd$

Oracle $\textsc{RatchetO}(i, upd)$
46 if $s_i \overset{?}{=} c_i$ then
47     return false
48 $(Y, \sigma) \leftarrow upd$
49 $\mathrm{forge} \leftarrow \mathrm{op}[c_i] \overset{?}{\neq}$ “exp” and $upd \overset{?}{\neq} \mathrm{auth}[3-i][c_i]$
50 if $\mathrm{unchanged}_i$ and forge then
51     if $c_i < j$ then
52         return false
53     if $c_i \overset{?}{=} j$ then
54         if $\sigma \overset{?}{\neq} \mathrm{F.Ev}(fk_i, Y)$ then
55             return false
56         $\mathrm{bad} \leftarrow \mathrm{true}$
57         return false                                   $\boxed{\mathrm{G}^*_{0,j}, \mathrm{I}_j}$
58 if $\sigma \overset{?}{\neq} \mathrm{F.Ev}(fk_i, Y)$ then
59     return false
60 if $\mathrm{op}[c_i] \overset{?}{=}$ “exp”
61     $\mathrm{restricted}_i \leftarrow \mathrm{true}$
62 if $upd \overset{?}{=} \mathrm{auth}[3-i][c_i]$ then
63     $\mathrm{restricted}_i \leftarrow \mathrm{false}$
64     $\mathrm{unchanged}_i \leftarrow \mathrm{true}$
65 else
66     $\mathrm{unchanged}_i \leftarrow \mathrm{false}$
67 $h \leftarrow \mathrm{H.Ev}(hk, c_i \parallel Y^{x_i})$
68 if $c_i + 1 \overset{?}{=} j$ then
69     $h \leftarrow rand$                                $\boxed{\mathrm{I}_j}$
70 $k_i \leftarrow h[1 \ldots \mathrm{BRKE.kl}]$
71 $fk_i \leftarrow h[\mathrm{BRKE.kl}+1 \ldots \mathrm{BRKE.kl}+\mathrm{F.kl}]$
72 $x_i \leftarrow \bot$
73 $c_i \leftarrow c_i + 1$
74 return true
```

Fig. 7. Games $\mathrm{G}_{0,j}, \mathrm{G}^*_{0,j}, \mathrm{I}_j$ for the proof of Theorem 1

```
Games $G_1, G_2$
01 $b \xleftarrow{\$} \{0,1\}$
02 $\text{unchanged}_1 \leftarrow$ true
03 $\text{unchanged}_2 \leftarrow$ true
04 $g \xleftarrow{\$} \mathbb{G}^*$
05 $k \xleftarrow{\$} \{0,1\}^{\text{BRKE.kl}}$
06 $hk \xleftarrow{\$} \{0,1\}^{\text{H.kl}}$
07 $fk \xleftarrow{\$} \{0,1\}^{\text{F.kl}}$
08 $\text{hash}[0] \leftarrow k \,\|\, fk$
09 $k_1, k_2 \leftarrow k, k$
10 $fk_1, fk_2 \leftarrow fk, fk$
11 $stk_1 \leftarrow (g, hk)$
12 $stk_2 \leftarrow (g, hk)$
13 $s_1, s_2 \leftarrow 0, 0$
14 $c_1, c_2 \leftarrow 0, 0$
15 $x_1, x_2 \leftarrow \bot, \bot$
16 $b' \xleftarrow{\$} \mathcal{D}^{\text{BCastO},\text{RatchetO},\text{Exp},\text{Ch}}(stk_1, stk_2)$
17 return $b' \overset{?}{=} b$

Oracle Exp$(i)$
18 if $\text{op}[c_i] \overset{?}{=}$ "ch" then
19     return $\bot$
20 if $\text{op}[s_i] \overset{?}{=}$ "ch" then
21     return $\bot$
22 $\text{op}[c_i] \leftarrow$ "exp"
23 $\text{op}[s_i] \leftarrow$ "exp"
24 if $c_i \geq 1$ then
25     $(Y, \sigma) \leftarrow \text{recv}_i$
26     $h \leftarrow \text{H.Ev}\left(hk, c_i - 1 \,\|\, Y^{x_i^{\text{old}}}\right)$
27     $\text{hash}[c_i] \leftarrow h$
28     $k_i \leftarrow h[1 \ldots \text{BRKE.kl}]$
29     $fk_i \leftarrow h[\text{BRKE.kl} + 1 \ldots \text{BRKE.kl} + \text{F.kl}]$
30 return $\left(x_i^{\text{old}}, (c_i, x_i, fk_i), k_i\right)$

Oracle Ch$(i)$
31 if $\text{restricted}_i$ then
32     return $k_i$
33 if $\text{op}[c_i] \overset{?}{=}$ "exp" then
34     return $\bot$
35 if $\text{unchanged}_i$ then
36     $k_i \leftarrow \text{hash}[c_i][0 \ldots \text{BRKE.kl}]$
37 $\text{op}[c_i] \leftarrow$ "ch"
38 if $\text{rkey}[c_i] \overset{?}{=} \bot$ then
39     $\text{rkey}[c_i] \xleftarrow{\$} \{0,1\}^{\text{RKE.kl}}$
40 if $b \overset{?}{=} 1$ then
41     return $k_i$
42 return $\text{rkey}[c_i]$

Oracle BCastO$(i)$
43 if $s_i \overset{?}{\neq} c_i$ then
44     return $\bot$
45 if $\text{op}[c_i] \overset{?}{=} \bot$
46     $\text{op}[c_i] \leftarrow$ "ch"
47 $x_i \xleftarrow{\$} \mathbb{Z}_p$
48 $X \leftarrow g^{x_i}$
49 $x_i^{\text{old}} \leftarrow x_i$
50 if $\text{unchanged}_i$ then
51     $fk_i \leftarrow \text{hash}[c_i][\text{BRKE.kl} + 1 \ldots \text{BRKE.kl} + \text{F.kl}]$
52 $\sigma \leftarrow \text{F.Ev}(fk_i, X)$
53 $upd \leftarrow (X, \sigma)$
54 $\text{auth}[i][c_i] \leftarrow upd$
55 $s_i \leftarrow s_i + 1$
56 return $upd$

Oracle RatchetO$(i, upd)$
57 if $s_i \overset{?}{=} c_i$ then
58     return false
59 $(Y, \sigma) \leftarrow upd$
60 forge $\leftarrow \text{op}[c_i] \overset{?}{\neq}$ "exp" and $upd \overset{?}{\neq} \text{auth}[3-i][c_i]$
61 if $\text{unchanged}_i$ and forge then
62     return false
63 if $\sigma \overset{?}{\neq} \text{F.Ev}(fk_i, Y)$ then
64     return false
65 $\text{recv}_i \leftarrow upd$
66 if $\text{op}[c_i] \overset{?}{=}$ "exp"
67     $\text{restricted}_i \leftarrow$ true
68 if $upd \overset{?}{=} \text{auth}[3-i][c_i]$ then
69     $\text{restricted}_i \leftarrow$ false
70     $\text{unchanged}_i \leftarrow$ true
71     if $\text{hash}[c_i + 1] \overset{?}{=} \bot$ then
72         $\text{hash}[c_i + 1] \leftarrow \text{H.Ev}(hk, c_i \,\|\, Y^{x_i})$        $\boxed{G_1}$
73         $\text{hash}[c_i + 1] \leftarrow \{0,1\}^{\text{H.ol}}$        $\boxed{G_2}$
74 else
75     $\text{unchanged}_i \leftarrow$ false
76     $h \leftarrow \text{H.Ev}(hk, c_i \,\|\, Y^{x_i})$
77     $k_i \leftarrow h[1 \ldots \text{BRKE.kl}]$
78     $fk_i \leftarrow h[\text{BRKE.kl} + 1 \ldots \text{BRKE.kl} + \text{F.kl}]$
79 $x_i \leftarrow \bot$
80 $c_i \leftarrow c_i + 1$
81 return true
```

Fig. 8. Games G_1, G_2 for the proof of Theorem 1

Reduction $\mathcal{O}_1^{\textsc{Up},\textsc{Ch},\textsc{Exp},\textsc{Hash}}(hk, g)$

```
$\mathrm{j}' \xleftarrow{\$} \{1, \ldots, Q_{\textsc{RatchetO}}\}$
$\mathrm{b} \xleftarrow{\$} \{0,1\}$
$\mathrm{b}' \leftarrow 0$
$\mathrm{exponent}_1 \leftarrow$ “x”
$\mathrm{exponent}_2 \leftarrow$ “y”
$\mathrm{unchanged}_1 \leftarrow \mathrm{true}$
$\mathrm{unchanged}_2 \leftarrow \mathrm{true}$
$rand \xleftarrow{\$} \{0,1\}^{\mathrm{H.ol}}$
$g \xleftarrow{\$} \mathbb{G}^*$
$k \xleftarrow{\$} \{0,1\}^{\mathrm{BRKE.kl}}$
$fk \xleftarrow{\$} \{0,1\}^{\mathrm{F.kl}}$
$k_1, k_2 \leftarrow k, k$
$fk_1, fk_2 \leftarrow fk, fk$
$stk_1 \leftarrow (g, hk)$
$stk_2 \leftarrow (g, hk)$
$c_1, c_2 \leftarrow 0, 0$
$x_1, x_2 \leftarrow \perp, \perp$
$b' \xleftarrow{\$} \mathcal{D}^{\mathrm{BCastO},\mathrm{RatchetO},\mathrm{ExpSim},\mathrm{ChSim}}(stk_1, stk_2)$
return $b' \overset{?}{=} b$
```

Oracle $\textsc{ExpSim}(i)$

```
if $\mathrm{op}[c_i] \overset{?}{=}$ “ch” then
    return $\perp$
if $\mathrm{op}[s_i] \overset{?}{=}$ “ch” then
    return $\perp$
if $\mathrm{op}[s_i] \overset{?}{=} \perp$ then
    $x, y \leftarrow \textsc{Exp}$
    $\mathrm{secret}[s_i][1] \leftarrow x$
    $\mathrm{secret}[s_i][2] \leftarrow y$
$\mathrm{op}[c_i] \leftarrow$ “exp”
$\mathrm{op}[s_i] \leftarrow$ “exp”
$r_i \leftarrow \mathrm{secret}[s_i][i]$
$x_i \leftarrow r_i$
if $c_i \overset{?}{=} s_i$ then
    $x_i \leftarrow \perp$
return $(r_i, (c_i, x_i, fk_i), k_i)$
```

Oracle $\textsc{ChSim}(i)$

```
if $\mathrm{restricted}_i$ then
    return $k_i$
if $\mathrm{op}[c_i] \overset{?}{=}$ “exp” then
    return $\perp$
$\mathrm{op}[c_i] \leftarrow$ “ch”
if $\mathrm{rkey}[c_i] \overset{?}{=} \perp$ then
    $\mathrm{rkey}[c_i] \xleftarrow{\$} \{0,1\}^{\mathrm{RKE.kl}}$
if $b \overset{?}{=} 1$ then
    return $k_i$
return $\mathrm{rkey}[c_i]$
```

Oracle $\mathrm{B}\textsc{Cast}\mathrm{O}(i)$

```
if $s_i \overset{?}{\neq} c_i$ then
    return $\perp$
if $\mathrm{op}[c_i] \overset{?}{=} \perp$
    $\mathrm{op}[c_i] \leftarrow$ “ch”
if $\mathrm{elem}[c_i] \overset{?}{=} \perp$ then
    $X, Y \xleftarrow{\$} \textsc{Up}$
    $\mathrm{elem}[c_i][1] \leftarrow X$
    $\mathrm{elem}[c_i][2] \leftarrow Y$
$X \leftarrow \mathrm{elem}[c_i][i]$
$\sigma \leftarrow \mathrm{F.Ev}(fk_i, X)$
$upd \leftarrow (X, \sigma)$
$\mathrm{auth}[i][c_i] \leftarrow upd$
$s_i \leftarrow s_i + 1$
return $upd$
```

Oracle $\textsc{RatchetO}(i, upd)$

```
if $s_i \overset{?}{=} c_i$ then
    return false
$(Y, \sigma) \leftarrow upd$
$\mathrm{forge} \leftarrow \mathrm{op}[c_i] \overset{?}{\neq}$ “exp” and $upd \overset{?}{\neq} \mathrm{auth}[3-i][c_i]$
if $\mathrm{unchanged}_i$ and forge then
    if $c_i < j$ then
        return false
    if $c_i \overset{?}{=} j$ then
        if $\sigma \overset{?}{\neq} \mathrm{F.Ev}(fk_i, Y)$ then
            return false
        $\mathrm{bad} \leftarrow \mathrm{true}$
        $b' \leftarrow 1$
        return false
if $\sigma \overset{?}{\neq} \mathrm{F.Ev}(fk_i, Y)$ then
    return false
if $\mathrm{op}[c_i] \overset{?}{=}$ “exp”
    $\mathrm{restricted}_i \leftarrow \mathrm{true}$
if $upd \overset{?}{=} \mathrm{auth}[3-i][c_i]$ then
    $\mathrm{restricted}_i \leftarrow \mathrm{false}$
    $\mathrm{unchanged}_i \leftarrow \mathrm{true}$
else
    $\mathrm{unchanged}_i \leftarrow \mathrm{false}$
if $c_i + 1 \overset{?}{\neq} j$ then
    $h \leftarrow \textsc{Hash}(c_i, Y, \mathrm{exponent}_i)$
else
    $h \xleftarrow{\$} \textsc{Ch}$
$k_i \leftarrow h[1 \ldots \mathrm{BRKE.kl}]$
$fk_i \leftarrow h[\mathrm{BRKE.kl} + 1 \ldots \mathrm{BRKE.kl} + \mathrm{F.kl}]$
$x_i \leftarrow \perp$
$c_i \leftarrow c_i + 1$
return true
```

Fig. 9. Reduction $\mathcal{O}_1$ for the proof of Theorem 1

```
Reduction $\mathcal{F}^{\text{Tag},\text{Verify}}$
$\text{j}' \xleftarrow{\$} \{0, \ldots, Q_{\text{RatchetO}}\}$
$\text{b} \xleftarrow{\$} \{0, 1\}$
$\text{unchanged}_1 \leftarrow \text{true}$
$\text{unchanged}_2 \leftarrow \text{true}$
$rand \xleftarrow{\$} \{0,1\}^{\text{H.ol}}$
$g \xleftarrow{\$} \mathbb{G}^*$
$k \xleftarrow{\$} \{0,1\}^{\text{BRKE.kl}}$
$hk \xleftarrow{\$} \{0,1\}^{\text{H.kl}}$
$fk \xleftarrow{\$} \{0,1\}^{\text{F.kl}}$
$k_1, k_2 \leftarrow k, k$
$fk_1, fk_2 \leftarrow fk, fk$
$stk_1 \leftarrow (g, hk)$
$stk_2 \leftarrow (g, hk)$
$c_1, c_2 \leftarrow 0, 0$
$x_1, x_2 \leftarrow \perp, \perp$
$b' \xleftarrow{\$} \mathcal{D}^{\text{BCastO},\text{RatchetO},\text{Exp},\text{Ch}}(stk_1, stk_2)$

Oracle Exp $(i)$
if $\text{op}[c_i] \overset{?}{=}$ "ch" then
    return $\perp$
if $\text{op}[s_i] \overset{?}{=}$ "ch" then
    return $\perp$
$\text{op}[c_i] \leftarrow$ "exp"
$\text{op}[s_i] \leftarrow$ "exp"
return $(x_i, (c_i, x_i, fk_i), k_i)$

Oracle Ch $(i)$
if $\text{restricted}_i$ then
    return $k_i$
if $\text{op}[c_i] \overset{?}{=}$ "exp" then
    return $\perp$
$\text{op}[c_i] \leftarrow$ "ch"
if $\text{rkey}[c_i] \overset{?}{=} \perp$ then
    $\text{rkey}[c_i] \xleftarrow{\$} \{0,1\}^{\text{RKE.kl}}$
if $b \overset{?}{=} 1$ then
    return $k_i$
return $\text{rkey}[c_i]$

Oracle BCastO $(i)$
if $s_i \overset{?}{\neq} c_i$ then
    return $\perp$
if $\text{op}[c_i] \overset{?}{=} \perp$
    $\text{op}[c_i] \leftarrow$ "ch"
$x_i \xleftarrow{\$} \mathbb{Z}_p$
$X \leftarrow g^{x_i}$
if $c_i \overset{?}{=} j'$ then
    $\sigma \leftarrow \text{Tag}(X)$
else
    $\sigma \leftarrow \text{F.Ev}(fk_i, X)$
$upd \leftarrow (X, \sigma)$
$\text{auth}[i][c_i] \leftarrow upd$
$s_i \leftarrow s_i + 1$
return $upd$

Oracle RatchetO $(i, upd)$
if $s_i \overset{?}{=} c_i$ then
    return false
$(Y, \sigma) \leftarrow upd$
$\text{forge} \leftarrow \text{op}[c_i] \overset{?}{\neq}$ "exp" and $upd \overset{?}{\neq} \text{auth}[3-i][c_i]$
if $\text{unchanged}_i$ and forge then
    if $c_i < j'$ then
        return false
    if $c_i \overset{?}{=} j'$ then
        if not $\text{Verify}(Y, \sigma)$ then
            return false
        $\text{bad} \leftarrow \text{true}$
        return false
if $c_i \overset{?}{=} j'$ then
    if not $\text{Verify}(Y, \sigma)$ then
        return false
else
    if $\sigma \overset{?}{\neq} \text{F.Ev}(fk_i, Y)$ then
        return false
if $\text{op}[c_i] \overset{?}{=}$ "exp"
    $\text{restricted}_i \leftarrow \text{true}$
if $upd \overset{?}{=} \text{auth}[3-i][c_i]$ then
    $\text{restricted}_i \leftarrow \text{false}$
    $\text{unchanged}_i \leftarrow \text{true}$
else
    $\text{unchanged}_i \leftarrow \text{false}$
$h \leftarrow \text{H.Ev}(hk, c_i \parallel Y^x)$
if $c_i + 1 \overset{?}{=} j'$ then
    $h \leftarrow rand$
$k_i \leftarrow h[1 \ldots \text{BRKE.kl}]$
$fk_i \leftarrow h[\text{BRKE.kl} + 1 \ldots \text{BRKE.kl} + \text{F.kl}]$
$x_i \leftarrow \perp$
$c_i \leftarrow c_i + 1$
return true
```

Fig. 10. Reduction $\mathcal{F}$ for the proof of Theorem 1

Reduction $\mathcal{O}_2^{\text{Up},\text{Ch},\text{Exp},\text{Hash}}(hk, g)$

```
$b \xleftarrow{\$} \{0,1\}$
$\text{exponent}_1 \leftarrow$ "x"
$\text{exponent}_2 \leftarrow$ "y"
$\text{unchanged}_1 \leftarrow \text{true}$
$\text{unchanged}_2 \leftarrow \text{true}$
$k \xleftarrow{\$} \{0,1\}^{\text{BRKE.kl}}$
$fk \xleftarrow{\$} \{0,1\}^{\text{F.kl}}$
$\text{hash}[0] \leftarrow k \,\|\, fk$
$k_1, k_2 \leftarrow k, k$
$fk_1, fk_2 \leftarrow fk, fk$
$stk_1 \leftarrow (g, hk)$
$stk_2 \leftarrow (g, hk)$
$s_1, s_2 \leftarrow 0, 0$
$c_1, c_2 \leftarrow 0, 0$
$b' \xleftarrow{\$} \mathcal{D}^{\text{BCastO},\text{RatchetO},\text{ExpSim},\text{ChSim}}(stk_1, stk_2)$
return $b' \overset{?}{=} b$
```

Oracle $\text{ExpSim}(i)$

```
if $\text{op}[c_i] \overset{?}{=}$ "ch" then
    return $\perp$
if $\text{op}[s_i] \overset{?}{=}$ "ch" then
    return $\perp$
if $c_i \geq 1$ then
    $(Y, \sigma) \leftarrow \text{recv}_i$
    $h \leftarrow \text{Hash}(c_i - 1, Y, \text{exponent}_i)$
    $\text{hash}[c_i] \leftarrow h$
    $k_i \leftarrow h[1 \ldots \text{BRKE.kl}]$
    $fk_i \leftarrow h[\text{BRKE.kl} + 1 \ldots \text{BRKE.kl} + \text{F.kl}]$
    if $\text{op}[s_i] \overset{?}{=} \perp$ then
        $x, y \leftarrow \text{Exp}$
        $\text{secret}[s_i][1] \leftarrow x$
        $\text{secret}[s_i][2] \leftarrow y$
    $r_i \leftarrow \text{secret}[s_i][i]$
    $x_i \leftarrow r_i$
    if $c_i \overset{?}{=} s_i$ then
        $x_i \leftarrow \perp$
$\text{op}[c_i] \leftarrow$ "exp"
$\text{op}[s_i] \leftarrow$ "exp"
return $(r_i, (c_i, x_i, fk_i), k_i)$
```

Oracle $\text{ChSim}(i)$

```
if $\text{restricted}_i$ then
    return $k_i$
if $\text{op}[c_i] \overset{?}{=}$ "exp" then
    return $\perp$
if $\text{op}[c_i] \overset{?}{=} \perp$ and $c_i \geq 1$ then
    $\text{hash}[c_i] \xleftarrow{\$} \text{Ch}$
$\text{op}[c_i] \leftarrow$ "ch"
if $\text{rkey}[c_i] \overset{?}{=} \perp$ then
    $\text{rkey}[c_i] \xleftarrow{\$} \{0,1\}^{\text{RKE.kl}}$
if $\text{unchanged}_i$ then
    $k_i \leftarrow \text{hash}[c_i][0 \ldots \text{BRKE.kl}]$
if $b \overset{?}{=} 1$ then
    return $k_i$
return $\text{rkey}[c_i]$
```

Oracle $\text{BCastO}(i)$

```
if $s_i \overset{?}{\neq} c_i$ then
    return $\perp$
if $\text{op}[c_i] \overset{?}{=} \perp$ then
    $\text{op}[c_i] \leftarrow$ "ch"
    $\text{hash}[c_i] \xleftarrow{\$} \text{Ch}$
if $\text{unchanged}_i$ then
    $fk_i \leftarrow \text{hash}[c_i][\text{BRKE.kl} + 1 \ldots \text{BRKE.kl} + \text{F.kl}]$
if $\text{elem}[c_i] \overset{?}{=} \perp$ then
    $X, Y \xleftarrow{\$} \text{Up}$
    $\text{elem}[c_i][1] \leftarrow X$
    $\text{elem}[c_i][2] \leftarrow Y$
$X \leftarrow \text{elem}[c_i][i]$
$\sigma \leftarrow \text{F.Ev}(fk_i, X)$
$upd \leftarrow (X, \sigma)$
$\text{auth}[i][c_i] \leftarrow upd$
$s_i \leftarrow s_i + 1$
return $upd$
```

Oracle $\text{RatchetO}(i, upd)$

```
if $s_i \overset{?}{=} c_i$ then
    return false
$(Y, \sigma) \leftarrow upd$
$\text{forge} \leftarrow \text{op}[c_i] \overset{?}{\neq}$ "exp" and $upd \overset{?}{\neq} \text{auth}[3 - i][c_i]$
if $\text{unchanged}_i$ and forge then
    return false
if $\sigma \overset{?}{\neq} \text{F.Ev}(fk_i, Y)$ then
    return false
$\text{recv}_i \leftarrow upd$
if $\text{op}[c_i] \overset{?}{=}$ "exp"
    $\text{restricted}_i \leftarrow \text{true}$
if $upd \overset{?}{=} \text{auth}[3 - i][c_i]$ then
    $\text{restricted}_i \leftarrow \text{false}$
    $\text{unchanged}_i \leftarrow \text{true}$
else
    $\text{unchanged}_i \leftarrow \text{false}$
    $h \leftarrow \text{Hash}(c_i, Y, \text{exponent}_i)$
    $k_i \leftarrow h[1 \ldots \text{BRKE.kl}]$
    $fk_i \leftarrow h[\text{BRKE.kl} + 1 \ldots \text{BRKE.kl} + \text{F.kl}]$
$c_i \leftarrow c_i + 1$
return true
```

Fig. 11. Reduction $\mathcal{O}_2$ for the proof of Theorem 1

References

1. Abdalla, M., Bellare, M., Rogaway, P.: DHIES: An encryption scheme based on the Diffie-Hellman problem. Contributions to IEEE P1363a (Sep 1998)
2. Abdalla, M., Bellare, M., Rogaway, P.: The oracle Diffie-Hellman assumptions and an analysis of DHIES. In: Naccache, D. (ed.) Topics in Cryptology – CT-RSA 2001. Lecture Notes in Computer Science, vol. 2020, pp. 143–158. Springer, Heidelberg, Germany, San Francisco, CA, USA (Apr 8–12, 2001). https://doi.org/10.1007/3-540-45353-9_12
3. Abe, M., Groth, J., Ohkubo, M., Tango, T.: Converting cryptographic schemes from symmetric to asymmetric bilinear groups. In: Garay, J.A., Gennaro, R. (eds.) Advances in Cryptology – CRYPTO 2014, Part I. Lecture Notes in Computer Science, vol. 8616, pp. 241–260. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 17–21, 2014). https://doi.org/10.1007/978-3-662-44371-2_14
4. Adrian, D., Bhargavan, K., Durumeric, Z., Gaudry, P., Green, M., Halderman, J.A., Heninger, N., Springall, D., Thomé, E., Valenta, L., VanderSloot, B., Wustrow, E., Zanella-Béguelin, S., Zimmermann, P.: Imperfect forward secrecy: How Diffie-Hellman fails in practice. In: Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security. p. 5–17. CCS '15, Association for Computing Machinery, New York, NY, USA (2015). https://doi.org/10.1145/2810103.2813707, https://doi.org/10.1145/2810103.2813707
5. Akinyele, J.A., Garman, C., Hohenberger, S.: Automating fast and secure translations from type-I to type-III pairing schemes. In: Ray, I., Li, N., Kruegel, C. (eds.) ACM CCS 2015: 22nd Conference on Computer and Communications Security. pp. 1370–1381. ACM Press, Denver, CO, USA (Oct 12–16, 2015). https://doi.org/10.1145/2810103.2813601
6. Balli, F., Rösler, P., Vaudenay, S.: Determining the core primitive for optimally secure ratcheting. In: Moriai, S., Wang, H. (eds.) Advances in Cryptology – ASIACRYPT 2020, Part III. Lecture Notes in Computer Science, vol. 12493, pp. 621–650. Springer, Heidelberg, Germany, Daejeon, South Korea (Dec 7–11, 2020). https://doi.org/10.1007/978-3-030-64840-4_21
7. Bellare, M., Rogaway, P.: The security of triple encryption and a framework for code-based game-playing proofs. In: Vaudenay, S. (ed.) Advances in Cryptology – EUROCRYPT 2006. Lecture Notes in Computer Science, vol. 4004, pp. 409–426. Springer, Heidelberg, Germany, St. Petersburg, Russia (May 28 – Jun 1, 2006). https://doi.org/10.1007/11761679_25
8. Bellare, M., Singh, A.C., Jaeger, J., Nyayapati, M., Stepanovs, I.: Ratcheted encryption and key exchange: The security of messaging. In: Katz, J., Shacham, H. (eds.) Advances in Cryptology – CRYPTO 2017, Part III. Lecture Notes in Computer Science, vol. 10403, pp. 619–650. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 20–24, 2017). https://doi.org/10.1007/978-3-319-63697-9_21
9. Borisov, N., Goldberg, I., Brewer, E.: Off-the-record communication, or, why not to use PGP. In: Proceedings of the 2004 ACM Workshop on Privacy in the Electronic Society. p. 77–84. WPES '04, Association for Computing Machinery, New York, NY, USA (2004). https://doi.org/10.1145/1029179.1029200, https://doi.org/10.1145/1029179.1029200
10. Durak, F.B., Vaudenay, S.: Bidirectional asynchronous ratcheted key agreement with linear complexity. In: Attrapadung, N., Yagi, T. (eds.) IWSEC 19: 14th International Workshop on Security, Advances in Information and Computer Security. Lecture Notes in Computer Science, vol. 11689, pp. 343–362. Springer, Heidelberg, Germany, Tokyo, Japan (Aug 28–30, 2019). https://doi.org/10.1007/978-3-030-26834-3_20
11. Durumeric, Z., Li, F., Kasten, J., Amann, J., Beekman, J., Payer, M., Weaver, N., Adrian, D., Paxson, V., Bailey, M., Halderman, J.A.: The matter of Heartbleed. In: Proceedings of the 2014 Conference on Internet Measurement Conference. p. 475–488. IMC '14, Association for Computing Machinery, New York, NY, USA (2014). https://doi.org/10.1145/2663716.2663755, https://doi.org/10.1145/2663716.2663755
12. Dutta, R., Barua, R., Sarkar, P.: Pairing-based cryptographic protocols : A survey. Cryptology ePrint Archive, Report 2004/064 (2004), https://eprint.iacr.org/2004/064
13. Galbraith, S., Paterson, K., Smart, N.: Pairings for cryptographers. Cryptology ePrint Archive, Report 2006/165 (2006), https://eprint.iacr.org/2006/165
14. Kiraz, M.S., Uzunkol, O.: Still wrong use of pairings in cryptography. Cryptology ePrint Archive, Report 2016/223 (2016), https://eprint.iacr.org/2016/223
15. Marlinspike, M.: WhatsApp's Signal protocol integration is now complete (2016), https://signal.org/blog/whatsapp-complete/ [Accessed: 2023-06-12]
16. Perrin, T., Marlinspike, M.: The double ratchet algorithm. Tech. rep., Open Whisper Systems (2016), https://signal.org/docs/specifications/doubleratchet/
17. Poettering, B., Rösler, P.: Asynchronous ratcheted key exchange. Cryptology ePrint Archive, Report 2018/296 (2018), https://eprint.iacr.org/2018/296

Acknowledgments Feedback and mentoring by Jiaxin Pan @ NTNU, LaTeX template by Felix Heuer @ RUB

A Reduction from IODHE to CDH in a Type 1 Pairing Group in the Random Oracle Model

A.1 Reduction from IODHE to ISCDHE in the Random Oracle Model

Lemma 1. *Given an adversary $\mathcal{A}$ against the* IODHE *security of* $\mathbb{G}, \mathrm{H}$ *in the random oracle model making* Q_{U} *queries to* Up*,* Q_{H} *queries to* Hash*,* Q_{E} *queries to* Exp*,* Q_{C} *queries to* Ch*, and* Q_{R} *queries to* RO*, there is an adversary $\mathcal{B}$ against the* ISCDHE *security of* $\mathbb{G}$ *such that*

$$\mathrm{Adv}^{\mathrm{IODHE}}_{\mathbb{G},\mathrm{H},\mathcal{A}} \leq \mathrm{Adv}^{\mathrm{ISCDHE}}_{\mathbb{G},\mathcal{B}} + \frac{Q_{\mathrm{H}}}{p}.$$

Adversary $\mathcal{B}$ makes at most Q_{U} *queries to its* ISCDHE Up *oracle,* Q_{E} *queries to its* ISCDHE Exp *oracle,* $Q_{\mathrm{R}}\,(2Q_{\mathrm{H}} + Q_{\mathrm{C}} + 1)$ *queries to its* ISCDHE DH1 *oracle,* $2Q_{\mathrm{R}}Q_{\mathrm{H}}$ *queries to its* ISCDHE DH2 *oracle, and* Q_{H}^2 *queries to its* ISCDHE DH3 *oracle.*

```
Game ISCDHE$_{\mathbb{G}}^{\mathcal{B}}$
    $g \xleftarrow{\$} \mathbb{G}^*$;  $v \leftarrow -1$
    $(j, Z) \xleftarrow{\$} \mathcal{B}^{\text{Up,Exp,DH1,DH2,DH3}}(g)$
    valid $\leftarrow (0 \leq j \leq v)$ and $\left(\mathrm{op}[j] \overset{?}{\neq} \text{“exp”}\right)$
    return valid and $\left(Z \overset{?}{=} g^{x[j]y[j]}\right)$
Oracle Up
    $v \leftarrow v + 1$
    $x[v] \xleftarrow{\$} \mathbb{Z}_p$
    $y[v] \xleftarrow{\$} \mathbb{Z}_p$
    return $\left(g^{x[v]}, g^{y[v]}\right)$
Oracle DH1 $(X, Z, i)$
    return $X^{y[i]} \overset{?}{=} Z$
Oracle DH2 $(Y, Z, i)$
    return $Y^{x[i]} \overset{?}{=} Z$
Oracle DH3 $(X, Y, i)$
    return $X^{y[i]} \overset{?}{=} Y^{x[i]}$
Oracle Exp
    $\mathrm{op}[v] \leftarrow$ “exp”
    return $(x[v], y[v])$
```

Fig. 12. The game defining the Independent Strong Computational Diffie-Hellman with Exposures (ISCDHE) assumption. The game is parametrized by a group $\mathbb{G}$. To win, the adversary $\mathcal{B}$ must compute $g^{x[j]y[j]}$ for some j where it did not query the Exp oracle.

We prove this lemma using an argument that closely follows the ODHE to SCDHE reduction given in Appendix A.1 in [8]. Let $\mathrm{IODHER}_{\mathbb{G},\mathrm{H},b^*}$ to be the IODHE game (defined in Figure 3) in the random oracle model, where the challenge bit b has been hard-coded to b^*, and the game returns 1 when $b' = 1$, where b' is the bit returned by $\mathcal{A}$. Note that in the random oracle model we introduce an additional oracle RO (z, κ) which is the hash function used by the scheme, modeled as a random function. The oracle returns a random string from $\{0,1\}^\kappa$ for each pair z, κ that is consistent across calls. We introduce 3 games:

Game G_0. This is $\mathrm{IODHE}_{\mathbb{G},\mathrm{H}}$ in the ROM where the challenge bit b has been hard-coded to equal 1, that is: $\mathrm{IODHER}_{\mathbb{G},\mathrm{H},1}$.

Game G_1. This is the same as G_0, but hash values are not kept consistent between the Ch and Hash oracles.

Game G_2. This is the same as G_1, but hash values are not kept consistent between the Ch oracle and the RO hash oracle.

Games $\mathrm{G}_0, \mathrm{G}_1$ and G_2 are given in Figure 13. Our goal is to transition from G_0 to G_2, and to prove that this is the transition from $\mathrm{IODHER}_{\mathbb{G},\mathrm{H},1}$ to $\mathrm{IODHER}_{\mathbb{G},\mathrm{H},0}$. We then wish to bound the difference of the games with the advantage of an adversary against ISCDHE. To complete the proof we use the following 4 claims.

(1) $\Pr[\mathrm{G}_0] = \Pr[\mathrm{IODHER}^{\mathcal{A}}_{\mathbb{G},\mathrm{H},1}]$

(2) $\Pr[\mathrm{G}_0] - \Pr[\mathrm{G}_1] \leq Q_{\mathrm{H}}/p$

(3) $\Pr[\mathrm{G}_1] - \Pr[\mathrm{G}_2] \leq \mathrm{Adv}^{\mathrm{ISCDHE}}_{\mathbb{G},\mathrm{H},\mathcal{B}}$ (For adversary $\mathcal{B}$ given in Figure 14)

(4) $\Pr[\mathrm{G}_2] = \Pr[\mathrm{IODHER}^{\mathcal{A}}_{\mathbb{G},\mathrm{H},0}]$

Using these claims we get the following in the random oracle model:

$$\begin{aligned}
\mathrm{Adv}^{\mathrm{IODHE}}_{\mathbb{G},\mathrm{H},\mathcal{A}} &= \Pr\left[\mathrm{IODHER}^{\mathcal{A}}_{\mathbb{G},\mathrm{H},1}\right] - \Pr\left[\mathrm{IODHER}^{\mathcal{A}}_{\mathbb{G},\mathrm{H},0}\right] \\
&= \Pr\left[\mathrm{G}_0{}^{\mathcal{A}}\right] - \Pr\left[\mathrm{G}_2{}^{\mathcal{A}}\right] \\
&= \left(\Pr\left[\mathrm{G}_0{}^{\mathcal{A}}\right] - \Pr\left[\mathrm{G}_1{}^{\mathcal{A}}\right]\right) + \left(\Pr\left[\mathrm{G}_1{}^{\mathcal{A}}\right] - \Pr\left[\mathrm{G}_2{}^{\mathcal{A}}\right]\right) \\
&\leq Q_{\mathrm{H}}/p + \mathrm{Adv}^{\mathrm{ISCDHE}}_{\mathbb{G},\mathcal{B}}.
\end{aligned}$$

The proofs of the 4 claims are based on those from [8]. The proofs of claims 1 and 4 are retold with little modification, while the proofs of claims 2 and 3 have been adapted to fit the new assumptions.

Claim (1). We modify $\mathrm{IODHER}_{\mathbb{G},\mathrm{H},1}$ to obtain G_0. This is done by replacing the accesses to the random oracle in the Ch, Hash and RO oracles with hash-tables, and ensuring consistency between them. This is done in each oracle by first checking if a hash value has been set in any of the two other tables, and propagating that value if it has. The need for the `mem` table is eliminated, and the value of r_1 is returned directly.

Claim (2). Transitioning from G_0 to G_1 we no longer maintain consistency between T_{ch} and T_{hash}. To bound the difference of the two games we use the fundamental theorem of game playing from [7] as the games are identical-until-bad with respect to the event $\mathrm{bad}_{0,1}$. Thus we have

$$\Pr\left[\mathrm{G}_0{}^{\mathcal{A}}\right] - \Pr\left[\mathrm{G}_1{}^{\mathcal{A}}\right] \leq \Pr\left[\mathrm{bad}^{\mathrm{G}_0{}^{\mathcal{A}}}_{0,1}\right], \tag{1}$$

where $\Pr\left[\mathrm{bad}^{\mathrm{G}_0{}^{\mathcal{A}}}_{0,1}\right]$ is the probability that the flag $\mathrm{bad}_{0,1}$ is set to true when G_0 is played by adversary $\mathcal{A}$.

Consider $\mathrm{bad}_{0,1}$ being set to `true` in Hash. For this to occur the adversary must query Hash with either $\left(i, g^{x[i]}, \text{“y”}\right)$ or $\left(i, g^{y[i]}, \text{“x”}\right)$ after $T_{\mathrm{ch}}\left[i, g^{x[i]y[i]}\right]$ has been set. For $T_{\mathrm{ch}}\left[i, g^{x[i]y[i]}\right]$ to be set, however, we must have $i \in S_{\mathrm{ch}}$, meaning that either possible call to Hash will return $\perp$, and that $\mathrm{bad}_{0,1}$ will not be set to `true`.

Now consider $\mathrm{bad}_{0,1}$ being set to `true` in Ch. For this to occur we must have neither $\left(v, g^{x[v]}, \text{“y”}\right) \in S_{\mathrm{hash}}$ nor $\left(v, g^{y[v]}, \text{“x”}\right) \in S_{\mathrm{hash}}$, but $T_{\mathrm{hash}}\left[v, g^{x[v]y[v]}\right]$ must have been set. This is only possible if the adversary made one of the hash queries *before* the counter v was incremented to its current value. Since the exponents $x[v]$ and $y[v]$ are sampled uniformly randomly at the same time as v is incremented in Up, any hash query has at most probability $1/p$ to correctly guess the future challenge value. Using the union bound over the queries to the Hash oracle, we obtain the bound of $\Pr\left[\mathrm{bad}^{\mathrm{G}_0{}^{\mathcal{A}}}_{0,1}\right] \leq Q_{\mathrm{H}}/p$.

Claim (3). Transitioning from G_1 to G_2 we no longer maintain consistency between T_{ch} and T. To bound the difference of the two games we use the fundamental theorem of game playing, as the games are identical-until-bad w.r.t the event $\mathrm{bad}_{1,2}$ and obtain

$$\Pr\left[\mathrm{G_1}^{\mathcal{A}}\right] - \Pr\left[\mathrm{G_2}^{\mathcal{A}}\right] \leq \Pr\left[\mathrm{bad}_{1,2}^{\mathrm{G_2}^{\mathcal{A}}}\right]. \tag{2}$$

For the event $\mathrm{bad}_{1,2}$ to be set in G_1 the adversary $\mathcal{A}$ must call Ch to set $T_{\mathrm{ch}}[v]$ and RO to set T with the corresponding value. This means that the adversary must be able to compute $g^{x[v]y[v]}$ and pass it to the random oracle. This enables a reduction to ISCDHE that checks every query to Ch and RO for a solution to the underlying ISCDHE instance. The reduction $\mathcal{B}$ is shown in Figure 14, and is based on the corresponding reduction given in [8], with modifications to fit the new assumptions. The command **abort** i, e is used to immidiately abort the execution of the reduction and return the index i and the challenge value e as the solution to the ISCDHE instance.

The reduction $\mathcal{B}$ simulates the view of adversary $\mathcal{A}$ in game G_2. While the implementation of oracles UpSim and ExpSim is trivial, care is taken to correctly manage the tables used in RO, Hash, and Ch. Since our reduction does not have access to the secrets $x[v]$ and $y[v]$ it cannot compute the group elements that are required as input to the hash function in Hash and Ch. This issue is trivial to solve in Ch, as there is exactly one group element $g^{x[v]y[v]}$ for each index v, and so using v to index the hash table instead is equivalent, as the mapping from v to $g^{x[v]y[v]}$ is injective. In oracle Hash we use the inputs i, X, `exponent` to index the hash table. We know that exponentiation by a constant is injective, but there are two options for the exponent, and queries with different values of `exponent` could map to the same group element and should therefore have the same hash. Due to this, our reduction must take care to synchronize the hash values for hash queries with different values of `exponent`.

In RO, the reduction extracts the values of hk, i and e to compare with previous queries. If the submission z does not decode into valid hk, i, e or the hash key or output length do not match those used by the scheme, no further checks are run. The reduction then checks for the presence of a challenge query at index i and, if one exists, uses oracle DH1 to check if $X_{\mathrm{cur}}[i]^{y[i]} = e$ and if the challenge query and this random oracle query should have the same hash value. If this is the case, the adversary has computed $g^{x[i]y[i]}$ and we immediately return the pair i, e using **abort**. If not, it then loops over every query made to the Hash oracle and uses either DH1 or DH2, depending on the value of `exponent`, to check if a hash value corresponding to i, e has already be computed. If such a query is found, the table T is updated with its value to keep RO in sync with Hash.

Oracle Hash uses logic similar to the last step in RO, but from a mirrored perspective, so it loops over every query to RO and uses oracles DH1 or DH2 to determine if it should have the same hash value as the current query to Hash. This keeps the values returned by Hash in sync with those returned by RO. Additionally, since the Hash oracle can accept queries for both exponents "x" and "y", it must make sure that queries like $(i, g^a, \text{"x"})$ and $\left(i, g^b, \text{"y"}\right)$ return the same hash value if $ax = by$. To do this, the reduction uses oracle DH3. Note that oracle Hash does not need to keep in sync with Ch, as our reduction is simulating G_2.

Oracle Ch uses logic similar to the first step in RO, but from a mirrored perspective, so it loops over every query (v, e) to RO and uses oracle DH1 to check if $X_{\mathrm{cur}}[v]^{y[v]} = e$ and if the random oracle query and this challenge query should have the same hash value.

If this is the case, the adversary has computed $g^{x[i]y[i]}$ and we immediately return the pair v, e using **abort**.

Our reduction $\mathcal{B}$ perfectly simultates the view of $\mathcal{A}$ in G_2 until $\mathrm{bad}_{1,2}$ is set to `true`, at which point the reduction immediately aborts and returns a tuple (i, e), which is a solution to the ISCDHE instance, and so $\mathrm{Adv}^{\mathrm{ISCDHE}}_{\mathbb{G},\mathrm{H},\mathcal{B}} = \Pr\left[\mathrm{bad}_{1,2}^{\mathrm{G}_2{}^{\mathcal{A}}}\right]$.

Claim (4). In G_2, hash values are consistent between RO and Hash, both of which are completely disconnected from hash values in Ch. This corresponds to the challenge bit b being set to 0 and hash values returned from Ch being random; the table T_{ch} taking the role of `mem` from $\mathrm{IODHER}_{\mathbb{G},\mathrm{H},0}$.

```
Games $G_0, G_1, G_2$
$hk \xleftarrow{\$} \{0,1\}^{\mathrm{H.kl}}$
$g \xleftarrow{\$} \mathbb{G}^*$
$v \leftarrow -1$
$b' \xleftarrow{\$} \mathcal{O}^{\mathrm{Up,Ch,Exp,Hash,RO}}(hk, g)$
return $b' \overset{?}{=} 1$

Oracle Up
$\mathrm{op} \leftarrow \varepsilon$
$v \leftarrow v + 1$
$x[v] \xleftarrow{\$} \mathbb{Z}_p$
$y[v] \xleftarrow{\$} \mathbb{Z}_p$
return $\left(g^{x[v]}, g^{y[v]}\right)$

Oracle Exp
if $\mathrm{op} \overset{?}{=}$ "ch" then
    return $\bot$
$\mathrm{op} \leftarrow$ "exp"
return $(x[v], y[v])$

Oracle RO $(z, \kappa)$
if $T[z,\kappa] \overset{?}{=} \bot$ then
    $T[z,\kappa] \xleftarrow{\$} \{0,1\}^{\mathrm{H.ol}}$
    $hk' \| i \| e \leftarrow z$
    if $(hk', \kappa) \overset{?}{=} (hk, \mathrm{H.ol})$ then
        if $T_{\mathrm{ch}}[i,e] \overset{?}{\neq} \bot$ then
            $\mathrm{bad}_{1,2} \leftarrow \mathrm{true}$
            $T[z,\kappa] \leftarrow T_{\mathrm{ch}}[i,e]$    $\boxed{G_0, G_1}$
        if $T_{\mathrm{hash}}[i,e] \overset{?}{\neq} \bot$ then
            $T[z,\kappa] \leftarrow T_{\mathrm{hash}}[i,e]$
return $T[z,\kappa]$

Oracle Ch
if $\left(\mathrm{op} \overset{?}{=} \text{"exp"}\right)$ or
    $\left(\left(v, g^{x[v]}, \text{"y"}\right) \in S_{\mathrm{hash}}\right)$ or
    $\left(\left(v, g^{y[v]}, \text{"x"}\right) \in S_{\mathrm{hash}}\right)$ then
    return $\bot$
$\mathrm{op} \leftarrow$ "ch"
$S_{\mathrm{ch}} \leftarrow S_{\mathrm{ch}} \cup \{v\}$
$e \leftarrow g^{x[v]y[v]}$
if $T_{\mathrm{ch}}[v] \overset{?}{=} \bot$ then
    $T_{\mathrm{ch}}[v] \xleftarrow{\$} \{0,1\}^{\mathrm{H.ol}}$
    if $T_{\mathrm{hash}}[v] \overset{?}{\neq} \bot$ then
        $\mathrm{bad}_{0,1} \leftarrow \mathrm{true}$
        $T_{\mathrm{ch}}[v] \leftarrow T_{\mathrm{hash}}[v]$    $\boxed{G_0}$
    if $T[hk \| v \| e, \mathrm{H.ol}] \overset{?}{\neq} \bot$ then
        $\mathrm{bad}_{1,2} \leftarrow \mathrm{true}$
        $T_{\mathrm{ch}}[v] \leftarrow T[hk \| v \| e, \mathrm{H.ol}]$    $\boxed{G_0, G_1}$
return $T_{\mathrm{ch}}[v]$

Oracle Hash $(i, X, \mathrm{exponent})$
if exponent $\overset{?}{=}$ "x" then
    if $i \in S_{\mathrm{ch}}$ and $X \overset{?}{=} g^{y[i]}$ then
        return $\bot$
    $e \leftarrow X^{x[v]}$
else if exponent $\overset{?}{=}$ "y" then
    if $i \in S_{\mathrm{ch}}$ and $X \overset{?}{=} g^{x[i]}$ then
        return $\bot$
    $e \leftarrow X^{y[v]}$
else
    return $\bot$
if $i \overset{?}{=} v$ then
    $S_{\mathrm{hash}} \leftarrow S_{\mathrm{hash}} \cup \{(i, X, \mathrm{exponent})\}$
if $T_{\mathrm{hash}}[i,e] \overset{?}{=} \bot$ then
    $T_{\mathrm{hash}}[i,e] \xleftarrow{\$} \{0,1\}^{\mathrm{H.ol}}$
    if $T_{\mathrm{ch}}[i,e] \overset{?}{\neq} \bot$ then
        $\mathrm{bad}_{0,1} \leftarrow \mathrm{true}$
        $T_{\mathrm{hash}}[i,e] \leftarrow T_{\mathrm{ch}}[i,e]$    $\boxed{G_0}$
    if $T[hk \| i \| e, \mathrm{H.ol}] \overset{?}{\neq} \bot$ then
        $T_{\mathrm{hash}}[i,e] \leftarrow T[hk \| i \| e, \mathrm{H.ol}]$
return $T_{\mathrm{hash}}[i,e]$
```

Fig. 13. Games $\mathrm{G}_0, \mathrm{G}_1, \mathrm{G}_2$ for the proof of Lemma 1.

```
Reduction $\mathcal{B}^{\textsc{Up},\mathrm{DH1},\mathrm{DH2},\mathrm{DH3},\textsc{Exp}}(g)$
$hk \xleftarrow{\$} \{0,1\}^{\mathrm{H.kl}}$
$v \leftarrow -1$
$\mathcal{A}^{\textsc{UpSim},\textsc{Ch},\textsc{ExpSim},\textsc{Hash},\mathrm{RO}}(hk, g)$
return $\perp, \perp$

Oracle UpSim
$\mathrm{op} \leftarrow \varepsilon$
$v \leftarrow v + 1$
$(X_{\mathrm{cur}}[v], Y_{\mathrm{cur}}[v]) \xleftarrow{\$} \textsc{Up}$
return $X_{\mathrm{cur}}[v], Y_{\mathrm{cur}}[v]$

Oracle ExpSim
if $\mathrm{op} \overset{?}{=}$ "ch" then
    return $\perp$
$\mathrm{op} \leftarrow$ "exp"
$(x, y) \leftarrow \textsc{Exp}$
return $x, y$

Oracle Ch
if $\left(\mathrm{op} \overset{?}{=} \text{"exp"}\right)$ or
        $((v, X_{\mathrm{cur}}[v], \text{"y"}) \in S_{\mathrm{hash}})$ or
        $((v, Y_{\mathrm{cur}}[v], \text{"x"}) \in S_{\mathrm{hash}})$ then
    return $\perp$
$\mathrm{op} \leftarrow$ "ch"
$S_{\mathrm{ch}} \leftarrow S_{\mathrm{ch}} \cup \{v\}$
if $T_{\mathrm{ch}}[v] \overset{?}{=} \perp$ then
    $T_{\mathrm{ch}}[v] \xleftarrow{\$} \{0,1\}^{\mathrm{H.ol}}$
    for $(i', e) \in S_{\mathrm{RO}}$ do
        if $i' \overset{?}{\neq} v$ then
            continue
        if $\mathrm{DH1}(X_{\mathrm{cur}}[v], e, v)$ then
            abort $i, e$
return $T_{\mathrm{ch}}[v]$

Oracle RO $(z, \kappa)$
if $T[z, \kappa] \overset{?}{=} \perp$ then
    $T[z, \kappa] \xleftarrow{\$} \{0,1\}^{\kappa}$
    $hk' \parallel i \parallel e \leftarrow z$
    if $(hk', \kappa) \overset{?}{=} (hk, \mathrm{H.ol})$ then
        $S_{\mathrm{RO}} \leftarrow S_{\mathrm{RO}} \cup \{(i, e)\}$
        if $i \in S_{\mathrm{ch}}$ then
            if $\mathrm{DH1}(X_{\mathrm{cur}}[i], e, i)$ then
                abort $i, e$
        for $(i', X', \mathrm{exponent}) \in S_{\mathrm{hash}}$ do
            if $i' \overset{?}{\neq} i$ then
                continue
            if $\mathrm{exponent} \overset{?}{=}$ "y" and $\mathrm{DH1}(X, e, i)$ then
                $T[z, \kappa] \leftarrow T_{\mathrm{hash}}[i, X, \mathrm{exponent}]$
            if $\mathrm{exponent} \overset{?}{=}$ "x" and $\mathrm{DH2}(X, e, i)$ then
                $T[z, \kappa] \leftarrow T_{\mathrm{hash}}[i, X, \mathrm{exponent}]$
return $T[z, \kappa]$

Oracle Hash $(i, X, \mathrm{exponent})$
if $\mathrm{exponent} \overset{?}{=}$ "x" then
    if $i \in S_{\mathrm{ch}}$ and $X \overset{?}{=} Y_{\mathrm{cur}}[i]$ then
        return $\perp$
else if $\mathrm{exponent} \overset{?}{=}$ "y" then
    if $i \in S_{\mathrm{ch}}$ and $X \overset{?}{=} X_{\mathrm{cur}}[i]$ then
        return $\perp$
else
    return $\perp$
if $i \overset{?}{=} v$ then
    $S_{\mathrm{hash}} \leftarrow S_{\mathrm{hash}} \cup \{(i, X, \mathrm{exponent})\}$
if $T_{\mathrm{hash}}[i, X, \mathrm{exponent}] \overset{?}{=} \perp$ then
    $T_{\mathrm{hash}}[i, X, \mathrm{exponent}] \xleftarrow{\$} \{0,1\}^{\mathrm{H.ol}}$
    for $(i', e) \in S_{\mathrm{RO}}$ do
        if $i' \overset{?}{\neq} i$ then
            continue
        if $\mathrm{exponent} \overset{?}{=}$ "y" and $\mathrm{DH1}(X, e, i)$ then
            abort $i, e$
        if $\mathrm{exponent} \overset{?}{=}$ "x" and $\mathrm{DH2}(X, e, i)$ then
            abort $i, e$
    for $(i', X', \mathrm{exponent}') \in S_{\mathrm{hash}}$ do
        if $i' \overset{?}{\neq} i$ then
            continue
        if $\mathrm{exponent} \overset{?}{=}$ "y" and $\mathrm{DH3}(X, X', i)$ then
            $T_{\mathrm{hash}}[i, X, \mathrm{exponent}] \leftarrow T_{\mathrm{hash}}[i', X', \mathrm{exponent}']$
        if $\mathrm{exponent} \overset{?}{=}$ "x" and $\mathrm{DH3}(X', X, i)$ then
            $T_{\mathrm{hash}}[i, X, \mathrm{exponent}] \leftarrow T_{\mathrm{hash}}[i', X', \mathrm{exponent}']$
return $T_{\mathrm{hash}}[i, X, \mathrm{exponent}]$
```

Fig. 14. Reduction $\mathcal{B}$ for the proof of Lemma 1.

A.2 Reduction from ISCDHE to ISCDH

Lemma 2. *Given an adversary $\mathcal{B}$ against the* ISCDHE *security of* $\mathbb{G}$ *making* Q_{U} *queries to* Up*,* Q_{E} *queries to* Exp*,* Q_1 *queries to* DH1*,* Q_2 *queries to* DH2*, and* Q_3 *queries to* DH3*, there is an adversary* $\mathcal{C}$ *against the* ISCDH *security of* $\mathbb{G}$ *such that*

$$\mathrm{Adv}^{\mathrm{ISCDHE}}_{\mathbb{G},\mathcal{B}} \leq Q_{\mathrm{U}}\mathrm{Adv}^{\mathrm{ISCDH}}_{\mathbb{G},\mathcal{C}}.$$

Adversary $\mathcal{C}$ *makes at most* Q_1 *queries to its* ISCDH DH1 *oracle,* Q_2 *queries to its* ISCDH DH2 *oracle, and* Q_3 *queries to its* ISCDH DH3 *oracle.*

```
Game ISCDH$^{\mathcal{C}}_{\mathbb{G}}$
01 $g \xleftarrow{\$} \mathbb{G}^*$
02 $x \xleftarrow{\$} \mathbb{Z}_p$
03 $y \xleftarrow{\$} \mathbb{Z}_p$
04 $Z \xleftarrow{\$} \mathcal{C}^{\mathrm{DH1,DH2,DH3}}(g, g^x, g^y)$
05 return $Z \overset{?}{=} g^{xy}$
```

```
Oracle DH1$(X, Z)$
06 return $X^y \overset{?}{=} Z$
Oracle DH2$(Y, Z)$
07 return $Y^x \overset{?}{=} Z$
Oracle DH3$(X, Y)$
08 return $X^y \overset{?}{=} Y^x$
```

Fig. 15. The game defining the Independent Strong Computational Diffie-Hellman (ISCDH) assumption. The game is parametrized by a group $\mathbb{G}$. To win, the adversary $\mathcal{C}$ must compute g^{xy}.

We prove this lemma using a simple guess-the-index attack, incurring a multiplicative security loss with factor Q_{U}. The authors of [8] give a reduction from the related assumption SCDHE to SCDH using rewinding and the random self-reducibility of CDH in Appendix A.1. With this they achieve a smaller, additive security loss instead. We note that this technique may be applicable to our ISCDHE to ISCDH reduction as well.

Our reduction $\mathcal{C}$ to the ISCDH security of group $\mathbb{G}$ is given in Figure 16. It makes a guess j' at the index adversary $\mathcal{B}$ will use for its submission, and simulates every other index by generating its own secrets $x[i]$ and $y[i]$. At index j', the reduction returns the ISCDH group elements X^* and Y^* from Up, delegates the calls to the DH oracles to the ISCDH DH oracles, and rejects calls to Exp. This perfectly simulates the view of $\mathcal{B}$ in the ISCDHE game when $j = j'$. When adversary $\mathcal{B}$ wins the ISCDHE game, we know that $Z = g^{x[j]y[j]}$, and as such our reduction $\mathcal{C}$ wins the ISCDH game when $j = j'$. Using this we get that $\mathrm{Adv}^{\mathrm{ISCDHE}}_{\mathbb{G},\mathcal{B}} \leq Q_{\mathrm{U}}\mathrm{Adv}^{\mathrm{ISCDH}}_{\mathbb{G},\mathcal{C}}$.

A.3 Reduction from ISCDH to CDH in a Type 1 Pairing Group

Lemma 3. *Given an adversary* $\mathcal{C}$ *against the* ISCDH *security of* $\mathbb{G}$ *making* Q_1 *queries to* DH1*,* Q_2 *queries to* DH2*, and* Q_3 *queries to* DH3*, there is an adversary* $\mathcal{D}$ *against the CDH security of a type 1 pairing group* $\mathbb{G}$ *such that*

$$\mathrm{Adv}^{\mathrm{ISCDH}}_{\mathbb{G},\mathcal{C}} = \mathrm{Adv}^{\mathrm{CDH}}_{\mathbb{G},\mathcal{D}}.$$

We prove this lemma by creating the trivial reduction $\mathcal{D}$ in Figure 17. The reduction perfectly simulates the different DH oracles by using the bilinear pairing e. The reduction returns the value submitted by $\mathcal{C}$, and wins the game if and only if $\mathcal{C}$ would have won its game. Therefore, $\mathrm{Adv}^{\mathrm{ISCDH}}_{\mathbb{G},\mathcal{C}} = \mathrm{Adv}^{\mathrm{CDH}}_{\mathbb{G},\mathcal{D}}$.

```
**Reduction** $\mathcal{C}^{\mathrm{DH1,DH2,DH3}}(g, X^*, Y^*)$
$v \leftarrow -1$
$j' \xleftarrow{\$} \{0, \dots, Q_{\mathrm{U}} - 1\}$
$(j, Z) \xleftarrow{\$} \mathcal{B}^{\textsc{Up},\textsc{Exp},\mathrm{DH1}\textsc{Sim},\mathrm{DH2}\textsc{Sim},\mathrm{DH3}\textsc{Sim}}(g)$
**return** $Z$

**Oracle** $\textsc{Up}$
$v \leftarrow v + 1$
**if** $v \overset{?}{=} j'$ **then**
    **return** $(X^*, Y^*)$
$x[v] \xleftarrow{\$} \mathbb{Z}_p$
$y[v] \xleftarrow{\$} \mathbb{Z}_p$
**return** $\left(g^{x[v]}, g^{y[v]}\right)$

**Oracle** $\textsc{Exp}$
**if** $v \overset{?}{=} j'$ **then**
    **abort**
$\mathrm{op}[v] \leftarrow$ "exp"
**return** $(x[v], y[v])$

**Oracle** $\mathrm{DH1}\textsc{Sim}(X, Z, i)$
**if** $i \overset{?}{=} j'$ **then**
    **return** $\mathrm{DH1}(X, Z)$
**return** $X^{y[i]} \overset{?}{=} Z$

**Oracle** $\mathrm{DH2}\textsc{Sim}(Y, Z, i)$
**if** $i \overset{?}{=} j'$ **then**
    **return** $\mathrm{DH2}(X, Z)$
**return** $Y^{x[i]} \overset{?}{=} Z$

**Oracle** $\mathrm{DH3}\textsc{Sim}(X, Y, i)$
**if** $i \overset{?}{=} j'$ **then**
    **return** $\mathrm{DH3}(X, Z)$
**return** $X^{y} \overset{?}{=} Y^{x}$
```

Fig. 16. Reduction $\mathcal{C}$ for the proof of Lemma 2.

This reduction essentially necessitates the use of pairing groups in our concrete scheme without any obvious purpose, other than being an artifact of the security proof. Pairing groups are utilized in many cryptographic schemes [12], but their application is occasionally problematic. As discussed in [13] and [14], the "black box" application of pairing groups in cryptographic schemes can sometimes cause issues related to efficiency, instanciability, and security. According to [14], a type 1 (symmetric) pairing with a large characteristic, p, would achieve instanciability and security for our scheme, at the cost of efficiency. We note that work has been done in regards to generic and automatic translation of schemes using type 1 pairings to schemes using the more efficient type 3 pairings [5] [3], but these are seen to be less efficient than a manual translation according to [14].

```
**Reduction** $\mathcal{D}(g, X^*, Y^*)$
$Z \xleftarrow{\$} \mathcal{C}^{\mathrm{DH1,DH2,DH3}}(g, X^*, Y^*)$
**return** $Z$

**Oracle** $\mathrm{DH1}(X, Z)$
**return** $e(X, Y^*) \overset{?}{=} e(Z, g)$

**Oracle** $\mathrm{DH2}(Y, Z)$
**return** $e(Y, X^*) \overset{?}{=} e(Z, g)$

**Oracle** $\mathrm{DH3}(X, Y)$
**return** $e(X, Y^*) \overset{?}{=} e(Y, X^*)$
```

Fig. 17. Reduction $\mathcal{D}$ for the proof of Lemma 3.