A preliminary version of this paper appears in CRYPTO 2017. This is the full version.

Ratcheted Encryption and Key Exchange: The Security of Messaging

Mihir Bellare[1] Asha Camper Singh[2] Joseph Jaeger[3]
Maya Nyayapati[4] Igors Stepanovs[5]

June 2017

Abstract

We aim to understand, formalize and provably achieve the goals underlying the core key-ratcheting technique of Borisov, Goldberg and Brewer, extensions of which are now used in secure messaging systems. We give syntax and security definitions for ratcheted encryption and key-exchange. We give a proven-secure protocol for ratcheted key exchange. We then show how to generically obtain ratcheted encryption from ratcheted key-exchange and standard encryption.

[1] Department of Computer Science & Engineering, University of California San Diego, 9500 Gilman Drive, La Jolla, California 92093, USA. Email: mihir@eng.ucsd.edu. URL: https://cseweb.ucsd.edu/~mihir/. Supported in part by NSF grants CNS-1526801 and CNS-1228890, ERC Project ERCC FP7/615074 and a gift from Microsoft.

[2] Salesforce. Work done while at UCSD. Email: acampersingh@gmail.com.

[3] Department of Computer Science & Engineering, University of California San Diego, 9500 Gilman Drive, La Jolla, California 92093, USA. Email: jsjaeger@eng.ucsd.edu. URL: https://cseweb.ucsd.edu/~jsjaeger/. Supported in part by grants of first author.

[4] Salesforce. Work done while at UCSD. Email: mnyayapa@gmail.com.

[5] Department of Computer Science & Engineering, University of California San Diego, 9500 Gilman Drive, La Jolla, California 92093, USA. Email: istepano@eng.ucsd.edu. URL: https://cseweb.ucsd.edu/~istepano/. Supported in part by grants of first author.

Contents

1 Introduction **2**

2 Preliminaries **5**

3 Oracle Diffie-Hellman with Exposures **7**

4 Ratcheted key exchange **9**
4.1 Definition of ratcheted key exchange 9
4.2 Security of ratcheted key exchange 12
4.3 Construction of a ratcheted key exchange scheme 14
4.4 Security proof for our ratcheted key exchange scheme 17

5 Ratcheted encryption **25**

A Oracle Diffie-Hellman with Exposures in ROM **32**
A.1 ODHE reduction to SCDH in ROM 34
A.2 SCDHE reduction to SCDH with rewinding 39

B Necessity of authenticaticating the update information **43**

1 Introduction

The classical view of cryptography was that the endpoints (Alice and Bob) are secure and the adversary is on the communication channel. The prevalence of malware and system vulnerabilities however makes endpoint compromise a serious and immediate threat. In their highly influential OTR (Off the Record) communication system, Borisov, Goldberg and Brewer (BGB) [9] attempt to mitigate the damage from endpoint compromise by regularly updating (ratcheting) the encryption key. (They do not call it ratcheting, this term originating later with Langley [18].) Ratcheting was then used by Open Whisper Systems in their Signal protocol [21], which in turn is used by WhatsApp and other secure messaging systems.

This widespread usage —WhatsApp alone reports handling 42 billion text messages *per day*— motivates an understanding and analysis of ratcheting: what is it aiming to accomplish, and does it succeed? The answer to this question does not seem clear. Indeed, in their SOK (Systemization of Knowledge) paper on secure messaging, UDBFPGS [24] survey many of the systems that existed at the time and attempt to classify them in terms of security, noting that security claims about ratcheting in different places include "forward-secrecy," "backward-secrecy," "self-healing" and "future secrecy," and concluding that *"The terms are controversial and vague in the literature"* [24, Section 2.D].

In this paper, we aim to formalize the goals that ratcheting appears to be targeting. We give definitions for ratcheted encryption and ratcheted key-exchange. We then give protocols (based on ones in use but not identical to them) to provably achieve the goals.

Our work aims to be selective rather than comprehensive. Our intent is to formalize and understand the simplest form of ratcheting that captures the essence of the goal, which is single, one-sided ratcheting. This (as we will see) is already complex enough. Extended forms of ratcheting are left as future work.

Ratcheting. The setting we consider is that sender Alice and receiver Bob hold keys $K_s = (k, \ldots)$ and $K_r = (k, \ldots)$, respectively, k representing a shared symmetric key and the ellipses indicating there may be more key information that may be party dependent. In practice, these keys are the result of a session-key exchange protocol that is authenticated either via the parties' certificates (TLS) or out-of-band (secure messaging), but ratcheting is about how these keys are used and updated, not about how they are obtained, and so we will not be concerned with the distribution method, instead viewing the initial keys as created and distributed by a trusted process.

In TLS, all data is secured under the shared key k with an authenticated encryption scheme. Under ratcheting, the key is constantly changing. As per BGB [9] it works roughly like this:

$$B \to A\colon\ g^{b_1}\ ;\ A \to B\colon\ g^{a_1},\ \mathcal{E}(k_1, M_1)\ ;\ B \to A\colon\ g^{b_2},\ \mathcal{E}(k_2, M_2)\ ;\ \ldots \tag{1}$$

Here a_i and b_i are random exponents picked by A and B respectively; $k_1 = H(k, g^{b_1 a_1})$, $k_2 = H(k_1, g^{a_1 b_2}), \ldots$; H is a hash function; $\mathcal{E}$ is an encryption function taking key and message to return a ciphertext; and g is the generator of an underlying group. Each party deletes its exponents and keys once they are no longer needed for encryption or decryption.

Contributions. This paper aims to lift ratcheting from a technique to a cryptographic primitive, with a precise syntax and formally-defined security goals. Once this is done, we specify and prove secure some protocols that are closely related to the in-use ones.

If ratcheting is to be a primitive, a syntax is the first requirement. As employed, the ratcheting technique is used within a larger protocol, and one has to ask what it might mean in isolation. To

allow a modular treatment, we decouple the creation of keys from their use, defining two primitives, ratcheted key exchange and ratcheted encryption. For each, we give a syntax. While ratcheting in apps is typically per message, our model is general and flexible, allowing the sender to ratchet the key at any time and encrypt as many messages as it likes under a given key before ratcheting again.

Next we give formal, game-based definitions of security for both ratcheted key exchange and ratcheted encryption. At the highest level, the requirement is that compromise (exposure in our model) revealing a party's current key and state should have only a local and temporary effect on security: a small hiccup, not compromising prior communications and after whose passage *both* privacy and integrity are somehow restored. This covers forward security (prior keys or communications remain secure) and backward security (future keys and communications remain secure). Amongst the issues in formalizing this is that following exposure there is some (necessary) time lag before security is regained, and that privacy and integrity are related. For ratcheted key exchange, un-exposed keys are required to be indistinguishable from random in the spirit of [5] —rather than merely, say, hard to recover— to allow them to be later securely used. For ratcheted encryption, the requirement is in the spirit of nonce-based authenticated encryption [22], so that authenticity in particular is provided.

The definitions are chosen to allow a modular approach to constructions. We exemplify by showing how to build ratcheted encryption generically from ratcheted key-exchange and multi-user-secure nonce-based encryption [7]. This allows us to focus on ratcheted key exchange.

We give a protocol for ratcheted key exchange that is based on DH key exchanges. The core technique is the same as in [9] and the in-use protocols, but there are small but important differences, including MAC-based authentication of the key-update values and the way keys are derived. We prove that our protocol meets our definition of ratcheted key exchange under the SCDH (Strong Computational Diffie-Hellman) assumption [1] in the random oracle model (ROM) [4]. The proof is obtained in two steps. The first is a standard-model reduction to an assumption we call ODHE (Oracle Diffie-Hellman with Exposures). The second is a validation of ODHE under SCDH in the ROM.

Model and syntax. Our syntax specifies a scheme RKE for ratcheted key exchange via three algorithms: initial key generation $\mathsf{RKE.IKg}$, sender key generation $\mathsf{RKE.SKg}$ and receiver key generation $\mathsf{RKE.RKg}$. See Fig. 3 for an illustration. The parties maintain output keys (representing the keys they are producing for an overlying application like ratcheted encryption) and session keys (local state for their internal use). At any time, the sender A can run $\mathsf{RKE.SKg}$ on its current keys to get update information *upd* that it sends to the receiver, as well as updated keys for itself. The receiver B correspondingly will run $\mathsf{RKE.RKg}$ on received update information and its current keys to get updated keys, transmitting nothing. $\mathsf{RKE.IKg}$ provides initial keys for the parties, what we called K_s and K_r above, that in particular contain an initial output key k (the same for both parties) and initial session keys. A ratcheted encryption scheme RE maintains the same three key-generation algorithms, now denoted $\mathsf{RE.IKg}$, $\mathsf{RE.SKg}$ and $\mathsf{RE.RKg}$, and adds an encryption algorithm $\mathsf{RE.Enc}$ for the sender —in the nonce-based vein [22], taking a key, nonce, message and header to deterministically return a ciphertext— and a corresponding decryption algorithm $\mathsf{RE.Dec}$ for the receiver. The key for encryption and decryption is what ratcheted key exchange referred to as the output key.

Besides a natural correctness requirement, we have a robustness requirement: if the receiver receives an update that it rejects, it maintains its state and will still accept a subsequent correct

update. This prevents a denial-of-service attack in which a single incorrect update sent to the receiver results in all future communications being rejected.

Security. In the spirit of BR [5] we give the adversary complete control of communication. Our definition of security for ratcheted key exchange in Section 4.2 is via a game KIND. After (trusted) initial key-generation, the game gives the adversary oracles to invoke either sender or receiver key generation and also to expose sender keys (both output and session). Roughly the requirement is that un-exposed keys be indistinguishable from random. The delicate issue is that this is true only under some conditions. Thus, exposure in one session will compromise the next session. Also, a post-expose active attack on the receiver (in which the adversary supplies the update information) can result in continued violation of integrity. Our game makes the necessary restrictions to capture these and other situations. For ratcheted encryption, the game RAE we give in Section 5 captures ratcheted authenticated encryption with nonce-based security. The additional oracles for the adversary are encryption and decryption. The requirement is that, for un-exposed and properly restricted keys, the adversary cannot distinguish whether its encryption and decryption oracles are real, or return random ciphertexts and $\perp$ respectively.

Schemes. Our ratcheted key exchange scheme in Section 4.3 is simple and efficient and uses the same basic DH technique as ratcheting in OTR [9] or WhatsApp, but analysis is quite involved. The sender's initial key includes g^b where b is part of the receiver's initial key, these quantities remaining static. Sender key generation algorithm RKE.SKg picks a random a and sends the update *upd* consisting of g^a *together with a mac under the prior session key* that is crucial to security. The output and next session key are derived via a hash function applied to g^{ab}. Theorem 4.1 establishes that the scheme meets our stringent notion of security for ratcheted key exchange. The proof uses a game sequence that includes a hybrid argument to reduce the security of the ratcheted key exchange to our ODHE (Oracle Diffie-Hellman with Exposures) assumption. The latter is an extension of the ODH assumption of [1] and, like the latter, can be validated in the ROM under the SCDH assumption of [1] (which in turn is a variant of the Gap-DH assumption of [20]). We show this in Appendix A. Ultimately, this yields a proof of security for our ratcheted key exchange protocol under the SCDH assumption in the ROM.

Our construction of a ratcheted encryption scheme in Section 5 is a generic combination of any ratcheted key exchange scheme (meeting our definition of security) and any nonce-based authenticated encryption scheme. Theorem 5.1 establishes that the scheme meets our notion of security for ratcheted encryption. The analysis is facilitated by assuming multi-user security for the base nonce-based encryption scheme as defined in [7], but a hybrid argument reduces this to the standard single-user security defined in [22]. Encryption schemes meeting this notion are readily available.

Setting and discussion. There are many variants of ratcheting. What we treat is one-sided ratcheting. This means one party (Alice) is a sender and the other (Bob) a receiver, rather than both playing both roles. In our model, compromises (exposures) are allowed only on the sender, not on the receiver. In particular the receiver has a static secret key whose compromise will immediately violate privacy of our schemes, regardless of updates. From the application perspective, our model and schemes are suitable for settings where the sender (for example a smartphone) is vulnerable to compromise but the receiver (for example a server with hardware-protected storage) can keep keys safely. In two-sided ratcheting, both the sender and the receiver may be compromised. Another dimension is single (what we treat) versus double ratcheting. In the latter, keys are also locally ratcheted via a forward-secure pseudorandom generator [8]. Conceptually, we decided to focus on

the single, one-sided case to keep definitions (already quite complex) as simple as possible while capturing the essence of the goal and method. But we note that what Signal implements, and what is thus actually used, is double, two-sided ratcheting. Treating this does not seem like a simple extension of what we do and is left as future work.

Secure Internet communication protocols (both TLS and messaging) start with a session-key exchange that provides session keys, K_s for the sender and K_r for the receiver. These are our initial keys, the starting points for ratcheting. These keys are not to be confused with higher-level, long-lived signing or other keys that are certified either explicitly (TLS) or out-of-band (messaging) and used for authentication in the session-key exchange.

Messaging sessions tend to be longer lived than typical TLS sessions, with conversations that are on-going for months. This is part of why messaging security seeks, via ratcheting, fine-grained forward and backward security. Still, exactly what threat ratcheting prevents in practice needs careful consideration. If the threat is malware on a communicant's phone that can directly exfiltrate text of conversations, ratcheting will not help. Ratcheting will be of more help when users delete old messages, when the malware is exfiltrating keys rather than text, and when its presence on the phone is limited through software security.

Related work. In concurrent and independent work, Cohn-Gordon, Cremers, Dowling, Garratt and Stebila (CCDGS) [10] give a formal analysis of the Signal protocol. The protocol they analyze includes ratcheting steps but stops at key distribution: unlike us, they do not consider, define or achieve ratcheted encryption. They treat Signal as a multi-stage session-key exchange protocol [17] in the tradition of authenticated session-key exchange [5, 3], with multiple parties and sessions. We instead consider ratcheted key exchange as a two-party protocol based on a trusted initial key distribution. This isolates ratcheted key exchange from the session key exchange used to produce the initial keys and allows a more modular treatment. They prove security (like us, in the ROM) under the Gap-DH [20] assumption while we prove it under the weaker SCDH [1] assumption. Ultimately their work and ours have somewhat different goals. Theirs is to analyze the particular Signal protocol. Ours is to isolate the core ratcheting method (as one of the more novel elements of the protocol) and formalize primitives reflecting its goals in the simplest possible way.

Cohn-Gordon, Cremers and Garratt (CCG) [11] study and compare different kinds of post-compromise security in contexts including authenticated key exchange. They mention ratcheting as a technique for maintaining security in the face of compromise.

Key-insulated cryptography [12, 13, 14] also targets forward and backward security but in a model where there is a trusted helper and an assumed-secure channel from helper to user that is employed to update keys. Implementing the secure channel is problematic due to the exposures [2]. Ratcheting in contrast works in a model where all communication is under adversary control.

2 Preliminaries

Notation and conventions. Let $\mathbb{N} = \{0, 1, 2, \ldots\}$ be the set of non-negative integers. Let ε denote the empty string. If $x \in \{0,1\}^*$ is a string then $|x|$ denotes its length, $x[i]$ denotes its i-th bit, and $x[i..j] = x[i] \ldots x[j]$ for $1 \leq i \leq j \leq |x|$. If mem is a table, we use $\mathsf{mem}[p]$ to denote the element of the table that is indexed by p. By $x \,\|\, y$ we denote a uniquely decodable concatenation of strings x and y (if lengths of x and y are fixed then $x \,\|\, y$ can be implemented using standard string concatenation). If X is a finite set, we let $x \leftarrow\$ X$ denote picking an element of X uniformly

Game $\mathrm{SUFCMA}_{\mathsf{F}}^{\mathcal{F}}$	Game $\mathrm{MAE}_{\mathsf{SE}}^{\mathcal{N}}$
$fk \leftarrow_{\$} \{0,1\}^{\mathsf{F.kl}}$ $\mathsf{win} \leftarrow \mathsf{false}$ $\mathcal{F}^{\textsc{Tag},\textsc{Verify}}$ Return win $\underline{\textsc{Tag}(m)}$ $\sigma \leftarrow \mathsf{F.Ev}(fk, m)$ $S \leftarrow S \cup \{(m,\sigma)\}$ Return σ $\underline{\textsc{Verify}(m,\sigma)}$ $\sigma' \leftarrow \mathsf{F.Ev}(fk, m)$ If $(\sigma = \sigma')$ and $((m,\sigma) \notin S)$ then $\mathsf{win} \leftarrow \mathsf{true}$ Return $(\sigma = \sigma')$	$b \leftarrow \{0,1\}$; $v \leftarrow 0$ $b' \leftarrow_{\$} \mathcal{N}^{\textsc{New},\textsc{Enc},\textsc{Dec}}$; Return $(b' = b)$ $\underline{\textsc{New}}$ $v \leftarrow v + 1$; $\mathsf{sk}[v] \leftarrow_{\$} \{0,1\}^{\mathsf{SE.kl}}$ $\underline{\textsc{Enc}(i,n,m,h)}$ If not $(1 \le i \le v)$ then return $\perp$ If $(i,n) \in U$ then return $\perp$ $c_1 \leftarrow \mathsf{SE.Enc}(\mathsf{sk}[i], n, m, h)$; $c_0 \leftarrow_{\$} \{0,1\}^{\mathsf{SE.cl}(\lvert m \rvert)}$ $U \leftarrow U \cup \{(i,n)\}$; $S \leftarrow S \cup \{(i,n,c_b,h)\}$ Return c_b $\underline{\textsc{Dec}(i,n,c,h)}$ If not $(1 \le i \le v)$ then return $\perp$ If $(i,n,c,h) \in S$ then return $\perp$ $m \leftarrow \mathsf{SE.Dec}(\mathsf{sk}[i], n, c, h)$ If $b = 1$ then return m else return $\perp$

Figure 1: Games defining strong unforgeability of function family F under chosen message attack, and multi-user authenticated encryption security of SE.

at random and assigning it to x. We use a special symbol $\perp$ to denote an empty table position, and we also return it as an error code indicating an invalid input; we assume that adversaries never pass $\perp$ as input to their oracles.

Algorithms may be randomized unless otherwise indicated. Running time is worst case. If A is an algorithm, we let $y \leftarrow A(x_1, \ldots; r)$ denote running A with random coins r on inputs $x_1, \ldots$ and assigning the output to y. We let $y \leftarrow_{\$} A(x_1, \ldots)$ be the result of picking r at random and letting $y \leftarrow A(x_1, \ldots; r)$. We let $[A(x_1, \ldots)]$ denote the set of all possible outputs of A when invoked with inputs $x_1, \ldots$. Adversaries are algorithms.

We use the code based game playing framework of [6]. (See Fig. 1 for an example.) We let $\Pr[\mathrm{G}]$ denote the probability that game G returns true. In code, uninitialized integers are assumed to be initialized to 0, Booleans to false, strings to the empty string, sets to the empty set, and tables are initially empty.

Function families. A family of functions F specifies a deterministic algorithm $\mathsf{F.Ev}$. Associated to F is a key length $\mathsf{F.kl} \in \mathbb{N}$, an input set $\mathsf{F.In}$, and an output length $\mathsf{F.ol}$. Evaluation algorithm $\mathsf{F.Ev}$ takes $fk \in \{0,1\}^{\mathsf{F.kl}}$ and an input $x \in \mathsf{F.In}$ to return an output $y \in \{0,1\}^{\mathsf{F.ol}}$.

Strong unforgeability under chosen message attack. Consider game SUFCMA of Fig. 1, associated to a function family F and an adversary $\mathcal{F}$. In order to win the game, adversary $\mathcal{F}$ has to produce a valid tag σ_{forge} for any message m_{forge}, satisfying the following requirement. The requirement is that $\mathcal{F}$ did not previously receive σ_{forge} as a result of calling its $\textsc{Tag}$ oracle with m_{forge} as input. The advantage of $\mathcal{F}$ in breaking the SUFCMA security of F is defined as $\mathsf{Adv}_{\mathsf{F},\mathcal{F}}^{\mathsf{sufcma}} = \Pr[\mathrm{SUFCMA}_{\mathsf{F}}^{\mathcal{F}}]$. If no adversaries can achieve a high advantage in breaking the SUFCMA security of F while using only bounded resources, we refer to F as a MAC algorithm and we refer to its key fk as a MAC key.

Symmetric encryption schemes. A symmetric encryption scheme SE specifies deterministic algo-

rithms $\mathsf{SE.Enc}$ and $\mathsf{SE.Dec}$. Associated to SE is a key length $\mathsf{SE.kl} \in \mathbb{N}$, a nonce space $\mathsf{SE.NS}$, and a ciphertext length function $\mathsf{SE.cl}\colon \mathbb{N} \to \mathbb{N}$. Encryption algorithm $\mathsf{SE.Enc}$ takes $sk \in \{0,1\}^{\mathsf{SE.kl}}$, a nonce $n \in \mathsf{SE.NS}$, a message $m \in \{0,1\}^*$ and a header $h \in \{0,1\}^*$ to return a ciphertext $c \in \{0,1\}^{\mathsf{SE.cl}(|m|)}$. Decryption algorithm $\mathsf{SE.Dec}$ takes sk, n, c, h to return message $m \in \{0,1\}^* \cup \{\perp\}$, where $\perp$ denotes incorrect decryption. Decryption correctness requires that $\mathsf{SE.Dec}(sk, n, \mathsf{SE.Enc}(sk, n, m, h), h) = m$ for all $sk \in \{0,1\}^{\mathsf{SE.kl}}$, all $n \in \mathsf{SE.NS}$, all $m \in \{0,1\}^*$, and all $h \in \{0,1\}^*$. Nonce-based symmetric encryption was introduced in [23], whereas [22] also considers it in the setting with associated data. In this work we consider only *nonce-based* symmetric encryption schemes *with associated data*; we omit repeating these qualifiers throughout the text, instead referring simply to "symmetric encryption schemes".

Multi-user authenticated encryption. Consider game MAE of Fig. 1, associated to a symmetric encryption scheme SE and an adversary $\mathcal{N}$. It extends the definition of *authenticated encryption with associated data* for nonce-based schemes [22] to the multi-user setting, first formalized in [7]. The adversary is given access to oracles New, Enc and Dec. It can increase the number of users by calling oracle New, which generates a new (secret) user key. For any of the user keys, the adversary can request encryptions of plaintext messages by calling oracle Enc and decryptions of ciphertexts by calling oracle Dec. In the real world (when $b = 1$), oracles Enc and Dec provide correct encryptions and decryptions. In the random world (when $b = 0$), oracle Enc returns uniformly random ciphertexts and oracle Dec returns the incorrect decryption symbol $\perp$. The goal of the adversary is to distinguish between these two cases. In order to avoid trivial attacks, $\mathcal{N}$ is not allowed to call Dec with ciphertexts that were returned by Enc. Likewise, we allow $\mathcal{N}$ to call Enc only once for every unique user-nonce pair (i, n). This can be strengthened to allow queries with repeated (i, n) and instead not allow queries with repeated (i, n, m, h), but the stronger requirement is satisfied by fewer schemes. The advantage of $\mathcal{N}$ in breaking the MAE security of SE is defined as $\mathsf{Adv}^{\mathsf{mae}}_{\mathsf{SE},\mathcal{N}} = 2\Pr[\mathrm{MAE}^{\mathcal{N}}_{\mathsf{SE}}] - 1$.

3 Oracle Diffie-Hellman with Exposures

The Oracle Diffie-Hellman (ODH) assumption [1] in a cyclic group requires that it is hard to distinguish between a random string and a hash function H applied to g^{xy}, even given g^x, g^y and an access to an oracle that returns $H(X^y)$ for arbitrary X (excluding $X = g^x$). We extend this assumption for multiple queries, based on a fixed g^y and arbitrarily many $g^{x[0]}, g^{x[1]}, \ldots$. For each index v we allow either to expose $x[v]$, or to get a challenge value; the challenge value is either a random string, or H applied to $g^{x[v]\cdot y}$. We also extend the hash function oracle to take a broader class of inputs.

Oracle Diffie-Hellman with Exposures assumption. Let $\mathbb{G}$ be a cyclic group of order $p \in \mathbb{N}$, and let $\mathbb{G}^*$ denote the set of its generators. Let H be a function family such that $\mathsf{H.In} = \{0,1\}^*$. Consider game ODHE of Fig. 2 associated to $\mathbb{G}, \mathsf{H}$ and an adversary $\mathcal{O}$, where $\mathcal{O}$ is required to call oracle Up at least once prior to making any oracle queries to Ch and Exp. The game starts by sampling a function key hk, a group generator g and a secret exponent y. The adversary is given hk, g, g^y and it has access to oracles Up, Ch, Exp, Hash. Oracle Up generates a new challenge exponent $x[v]$ and returns $g^{x[v]}$, where v is an integer counter that denotes the number of the current challenge exponent (indexed from 0) and is incremented by 1 at the start of every call to oracle Up. Oracle Hash takes an arbitrary integer i, an arbitrary string s and a group element X to return $\mathsf{H.Ev}(hk, i \,\|\, s \,\|\, X^y)$.

Game $\mathrm{ODHE}^{\mathcal{O}}_{\mathbb{G},\mathsf{H}}$

$b \leftarrow\$ \{0,1\}$; $hk \leftarrow\$ \{0,1\}^{\mathsf{H.kl}}$; $g \leftarrow\$ \mathbb{G}^*$; $y \leftarrow\$ \mathbb{Z}_p$; $v \leftarrow -1$
$b' \leftarrow\$ \mathcal{O}^{\textsc{Up},\textsc{Ch},\textsc{Exp},\textsc{Hash}}(hk, g, g^y)$; Return $(b' = b)$

Up

$\mathsf{op} \leftarrow \varepsilon$; $v \leftarrow v + 1$; $x[v] \leftarrow\$ \mathbb{Z}_p$; Return $g^{x[v]}$

Ch(s)

If (op = "exp") or $((v, s, g^{x[v]}) \in S_{\mathsf{hash}})$ then return $\perp$
$\mathsf{op} \leftarrow$ "ch" ; $S_{\mathsf{ch}} \leftarrow S_{\mathsf{ch}} \cup \{(v, s, g^{x[v]})\}$; $e \leftarrow g^{x[v]\cdot y}$
If $\mathsf{mem}[v, s, e] = \perp$ then $\mathsf{mem}[v, s, e] \leftarrow\$ \{0,1\}^{\mathsf{H.ol}}$
$r_1 \leftarrow \mathsf{H.Ev}(hk, v \,\|\, s \,\|\, e)$; $r_0 \leftarrow \mathsf{mem}[v, s, e]$; Return r_b

Exp

If op = "ch" then return $\perp$
$\mathsf{op} \leftarrow$ "exp" ; Return $x[v]$

Hash(i, s, X)

If $(i, s, X) \in S_{\mathsf{ch}}$ then return $\perp$
If $i = v$ then $S_{\mathsf{hash}} \leftarrow S_{\mathsf{hash}} \cup \{(i, s, X)\}$
Return $\mathsf{H.Ev}(hk, i \,\|\, s \,\|\, X^y)$

Figure 2: Game defining Oracle Diffie-Hellman with Exposures assumption for $\mathbb{G}, \mathsf{H}$.

For each counter value v, the adversary can choose to either call oracle Exp to get the value of $x[v]$ or call oracle Ch with input s to get a challenge value that is generated as follows. In the real world (when $b = 1$) oracle Ch returns $\mathsf{H.Ev}(hk, v \,\|\, s \,\|\, g^{x[v]\cdot y})$ and in the random world (when $b = 0$) it returns a uniformly random element from $\{0,1\}^{\mathsf{H.ol}}$. The goal of the adversary is to distinguish between these two cases. Oracle Ch can be called multiple times per challenge exponent, and it returns consistent outputs regardless of the challenge bit's value. The advantage of $\mathcal{O}$ in breaking the ODHE security of $\mathbb{G}, \mathsf{H}$ is defined as $\mathsf{Adv}^{\mathsf{odhe}}_{\mathbb{G},\mathsf{H},\mathcal{O}} = 2\Pr[\mathrm{ODHE}^{\mathcal{O}}_{\mathbb{G},\mathsf{H}}] - 1$.

In order to avoid trivial attacks, $\mathcal{O}$ is not allowed to query oracle Hash on input (i, s, X) if $X = g^{x[i]}$ and if oracle Ch was already called with input s when the counter value was $v = i$. Note that adversary is allowed to win the game if it happens to guess a future challenge exponent x and query it to oracle Hash ahead of time; the corresponding triple (i, s, X) will not be added to the set of inputs S_{hash} that are not allowed to be made to oracle Ch. Finally, recall that the string concatenation operator $\|$ is defined to produce uniquely decodable strings, which helps to avoid trivial string padding attacks.

Plausibility of the ODHE assumption. We do not know of any group $\mathbb{G}$ and function family H that can be shown to achieve ODHE in the standard model. The original ODH assumption of [1] was justified by a reduction in the random oracle model to the Strong Computational Diffie-Hellman (SCDH) assumption. The latter was defined in [1] and is a weaker version of the Gap Diffie-Hellman assumption from [20]. In Appendix A we give a definition for the SCDH assumption and prove that it also implies the ODHE assumption in the random oracle model.

We provide this result as a corollary of two lemmas. The lemmas use the Strong Computational Diffie-Hellman with Exposures (SCDHE) assumption as an intermediate step, where SCDHE is a novel assumption that extends SCDH to allow multiple challenge queries, and to allow exposures. To formalize our result, we define the Oracle Diffie-Hellman with Exposures in ROM (ODHER)

assumption that is equivalent to the ODHE assumption in the random oracle model.

The first lemma establishes that SCDHE implies ODHE in the random oracle model, by a reduction from ODHER to SCDHE. The proof of this lemma emulates the ODH to SCDH reduction of [1]. In their reduction, the SCDH adversary simulates the random oracle and the hash oracle for the ODH adversary; it uses its own decisional-DH oracle to check whether the ODH adversary feeds g^{xy} for the challenge values of x and y, and to maintain consistency between simulated oracle outputs. This consistency maintenance is the main source of complexity in our reduction because —in addition to the oracles mentioned above— we must also ensure that the simulated challenge oracle is consistent.

The second lemma is a standard model reduction from SCDHE to SCDH. This reduction is a standard "guess the index" reduction in which our SCDH adversary guesses which query the SCDHE adversary will attack. The SCDH adversary replaces the answer to this query with the challenge values it was given and replaces all other oracle queries with challenges that it has generated itself. As usual, this results in a multiplicative loss of security, so the final theorem (combining both lemmas) has a bound of the form $\mathsf{Adv}^{\mathsf{odher}}_{\mathbb{G},\mathsf{H},\mathcal{O}} \leq q_{\mathrm{UP}} \cdot \mathsf{Adv}^{\mathsf{scdh}}_{\mathbb{G},\mathcal{S}}$, where $\mathcal{S}$ is the SCDH adversary and q_{UP} is the number of UP queries made by ODHER adversary $\mathcal{O}$.

Because of the multiplicative loss of security caused by the second lemma we also examine the possibility of using Diffie-Hellman self-reducibility techniques to obtain a tighter bound on the reduction from SCDHE to SCDH. The possibility of exposures in SCDHE makes this much more difficult than one might immediately realize. We present a reduction that succeeds despite these difficulties, by using significantly more complicated methods than in our first example of this deduction. Specifically we build an SCDH adversary that makes guesses about the future behavior of the SCDHE adversary it was given, and "rewinds" this adversary whenever its guess was incorrect. Thus we ultimately obtain the much tighter bound of $\mathsf{Adv}^{\mathsf{odher}}_{\mathbb{G},\mathsf{H},\mathcal{O}} \leq \mathsf{Adv}^{\mathsf{scdh}}_{\mathbb{G},\mathcal{S}_u} + q_{\mathrm{UP}} \cdot 2^{-u}$. Here $\mathcal{S}_u$ is the SCDH adversary that is defined for any parameter $u \in \mathbb{N}$ that bounds its worst case running time, and q_{UP} is the number of UP queries made by ODHER adversary $\mathcal{O}$.

4 Ratcheted key exchange

Ratcheted key exchange allows users to agree on shared secret keys while providing very strong security guarantees. In this work we consider a setting that encompasses two parties, and we assume that only one of them sends key agreement messages. We call this party a sender, and the other party a receiver. This model enables us to make the first steps towards capturing the schemes that are used in the real world messaging applications. Future work could extend our model to allow both parties to send key agreement messages, and to consider the group chat setting where multiple users engage in shared conversations.

4.1 Definition of ratcheted key exchange

Consider Fig. 3 for an overview of algorithms that constitute a racheted key exchange scheme RKE, and the interaction between them. The algorithms are RKE.IKg, RKE.SKg and RKE.RKg. We will first provide an informal description of their functionality, and then formalize their syntax and correctness requirements.

Initial key generation algorithm RKE.IKg generates and distributes the following keys: $k, stk_s, stk_r, sek_s, sek_r$. Output key k is the initial shared secret key that can be used by both parties for any purpose such as running a symmetric encryption scheme. Static keys stk_s and stk_r are long-term

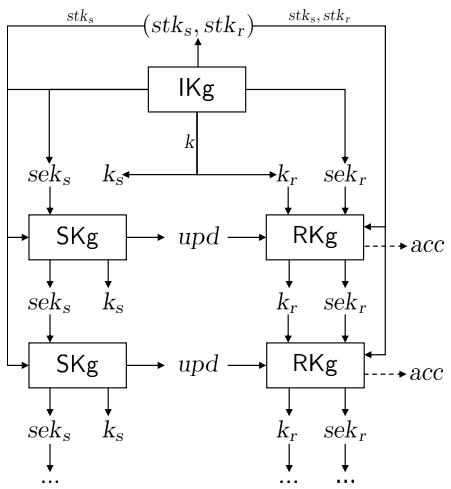

Figure 3: The interaction between ratcheted key exchange algorithms.

---

keys that will not get updated over time. It is assumed that stk_s is known to all parties, whereas stk_r contains potentially secret information and will be known only by the receiver. Session keys sek_s and sek_r contain secret information that is required for future key exchanges, such as MAC keys (to ensure the authenticity of key exchange) and temporary secrets (that could be used for the generation of the next output keys). As a result of running $\mathsf{RKE.IKg}$, the sender gets stk_s, sek_s, k_s and the receiver gets stk_s, stk_r, sek_r, k_r, where $k_s = k_r = k$. We use "s" and "r" as subscripts for output keys and session keys, to indicate that the particular key is owned by the sender or by the receiver, respectively. Note that normally both parties will have the same output key (i.e. $k_s = k_r$), but this might not be true if an attacker succeeds to tamper with the protocol.

Next we define sender's and receiver's key generation algorithms $\mathsf{RKE.SKg}$ and $\mathsf{RKE.RKg}$. These algorithms model the key ratcheting process that generates new session keys and output keys while deleting the corresponding old keys.

Sender's key generation algorithm $\mathsf{RKE.SKg}$ is run whenever the sender wants to produce a new shared secret key. It takes the sender's static key stk_s and the sender's session key sek_s. It returns an updated sender's session key sek_s, a new output key k_s, and update information upd. The update information is used by the receiver to generate the same output key.

Receiver's key generation algorithm $\mathsf{RKE.RKg}$ takes sender's static key stk_s, receiver's static key stk_r, receiver's session key sek_r, update information upd (received from the sender) and the current shared output key k_r. It returns receiver's session key sek_r, output key k_r, and a Boolean flag acc indicating whether the new keys were generated succesfully. Setting $acc = \mathsf{false}$ will generally mean that the received update information was rejected; our correctness definition will require that in such case the receiver's output key k_r and the receiver's session key sek_r should remain unchanged. This requirement is the reason why $\mathsf{RKE.RKg}$ takes the old value of k_r as one of its inputs.

Ratcheted key exchange schemes. A ratcheted key exchange scheme RKE specifies algorithms $\mathsf{RKE.IKg}$, $\mathsf{RKE.SKg}$ and $\mathsf{RKE.RKg}$. Associated to RKE is an output key length $\mathsf{RKE.kl} \in \mathbb{N}$ and sender's key generation randomness space $\mathsf{RKE.RS}$. Initial key generation algorithm $\mathsf{RKE.IKg}$ returns k, sek_s, (stk_s, stk_r, sek_r), where $k \in \{0,1\}^{\mathsf{RKE.kl}}$ is an output key, sek_s is a sender's session key, and stk_s, stk_r, sek_r are sender's static key, receiver's static key and receiver's session key, respectively. The sender's and receiver's output keys are initialized to $k_s = k_r = k$. Sender's key generation

Game RKE-COR$^{\mathcal{C}}_{\mathsf{R}}$	Game RE-COR$^{\mathcal{C}}_{\mathsf{R}}$
$\mathsf{bad} \leftarrow \mathsf{false}$ $(k, sek_s, (stk_s, stk_r, sek_r)) \leftarrow \mathsf{R.IKg}$ $k_s \leftarrow k$; $k_r \leftarrow k$ $\mathcal{C}^{\textsc{Up},\textsc{RatRec}}$; Return ($\mathsf{bad} = \mathsf{false}$)	$\mathsf{bad} \leftarrow \mathsf{false}$ $(k, sek_s, (stk_s, stk_r, sek_r)) \leftarrow \mathsf{R.IKg}$ $k_s \leftarrow k$; $k_r \leftarrow k$ $\mathcal{C}^{\textsc{Up},\textsc{RatRec},\textsc{Enc}}$; Return ($\mathsf{bad} = \mathsf{false}$)

$\textsc{Up}$

$r \leftarrow_{\$} \mathsf{R.RS}$; $(sek_s, k_s, upd) \leftarrow \mathsf{R.SKg}(stk_s, sek_s; r)$
$(sek_r, k_r, acc) \leftarrow_{\$} \mathsf{R.RKg}(stk_s, stk_r, sek_r, upd, k_r)$
If not $((acc = \mathsf{true})$ and $(k_s = k_r))$ then $\mathsf{bad} \leftarrow \mathsf{true}$

$\textsc{RatRec}(upd)$

$(sek'_r, k'_r, acc) \leftarrow_{\$} \mathsf{R.RKg}(stk_s, stk_r, sek_r, upd, k_r)$
If $(acc = \mathsf{false})$ and not $((k'_r = k_r)$ and $(sek'_r = sek_r))$ then $\mathsf{bad} \leftarrow \mathsf{true}$

$\textsc{Enc}(n, m, h)$

$c \leftarrow \mathsf{R.Enc}(k_s, n, m, h)$; $m' \leftarrow \mathsf{R.Dec}(k_r, n, c, h)$; If $(m' \neq m)$ then $\mathsf{bad} \leftarrow \mathsf{true}$

Figure 4: Game RKE-COR defining correctness of ratcheted key exchange scheme R, and game RE-COR defining correctness of ratcheted encryption scheme R. Oracles Up and RatRec are used in both games, whereas oracle Enc is only used in game RE-COR.

algorithm RKE.SKg takes stk_s, sek_s and randomness $r \in \mathsf{RKE.RS}$ to return a new sender's session key sek_s, a new sender's output key $k_s \in \{0,1\}^{\mathsf{RKE.kl}}$, and update information upd. Receiver's key generation algorithm RKE.RKg takes stk_s, stk_r, sek_r, upd and receiver's output key $k_r \in \{0,1\}^{\mathsf{RKE.kl}}$ to return a new receiver's session key sek_r, a new receiver's output key $k_r \in \{0,1\}^{\mathsf{RKE.kl}}$, and a flag $acc \in \{\mathsf{true}, \mathsf{false}\}$.

Correctness of ratcheted key exchange. Consider game RKE-COR of Fig. 4 associated to a ratcheted key exchange scheme R and an adversary $\mathcal{C}$, where $\mathcal{C}$ is provided with an access to oracles Up and RatRec.

Oracle Up runs algorithm R.SKg to generate a new sender's output key k_s along with the corresponding update information upd; it then runs R.RKg with upd as input to generate a new receiver's output key k_r. It is required that $acc = \mathsf{true}$ and $k_s = k_r$ at the end of every Up call. This means that if the receiver uses update information received from the sender (in the correct order), it is guaranteed to successfully generate the same output keys as the sender.

Oracle RatRec takes update information upd of adversary's choice and attempts to run R.RKg with upd (and current receiver's keys) as input. The correctness requires that if the receiver's key update fails (meaning $acc = \mathsf{false}$) then the receiver's keys k_r, sek_r remain unchanged. This means that if receiver's attempt to generate new keys is not successful (e.g. if the update information is corrupted in transition), then the receiver's key generation algorithm should not corrupt the receiver's current keys. This is a usability property that requires that it is possible to recover from failures, meaning that the receiver can later re-run its key generation algorithm with the correct update information to successfully produce its next pair of (session and output) keys.

We consider an unbounded adversary and allow it to call its oracles in any order. The advantage of $\mathcal{C}$ breaking the correctness of R is defined as $\mathsf{Adv}^{\mathsf{rkecor}}_{\mathsf{R},\mathcal{C}} = 1 - \Pr[\text{RKE-COR}^{\mathcal{C}}_{\mathsf{R}}]$. Correctness property requires that $\mathsf{Adv}^{\mathsf{rkecor}}_{\mathsf{R},\mathcal{C}} = 0$ for all unbounded adversaries $\mathcal{C}$. Note that our definition of the correctness game with an unbounded adversary is equivalent to a more common correctness definition that would instead explicitly quantify over all randomness choices of all algorithms. We

stress that our correctness definition does *not* require any security properties. In particular, it does not require that the update information is authenticated because oracle RatRec considers only the case when R.RKg sets $acc = \mathsf{false}$.

Our definition requires *perfect* correctness. However, it can be relaxed by requiring that adversary $\mathcal{C}$ can only make a bounded number of calls to its oracles, and further requiring that its advantage of winning the game is negligible.

4.2 Security of ratcheted key exchange

Ratcheted key exchange attempts to provide strong security guarantees even in the presence of an attacker that can steal the secrets stored by the sender. Specifically, we consider an active attacker that is able to intercept and modify any update information sent from the sender to the receiver. The goal is that the attacker cannot distinguish the produced output keys from random strings, and cannot make the two parties agree on output keys that do not match. Furthermore, we desire certain stronger security properties to hold even if the attacker manages to steal secrets stored by the sender, which we refer to as forward security and backward security. Forward security requires that such an attacker cannot distinguish prior keys from random. Backward security requires that the knowledge of sender's secrets at the current time period can not be used to distinguish keys generated (at some near point) in the future from random strings. Recall that our model is intentionally one-sided; exposure of receiver's secrets is not allowed. In particular, compromise of all of the receiver's secrets will permanently compromise security.

It is clear that if an attacker steals the secret information of the sender, then it can create its own update information resulting in the receiver agreeing on a "secret" key that is known by the attacker. It can be difficult to say what restrictions should be placed on the keys that the attacker makes the receiver agree to. Is it a further breach of security if the attacker then later causes the sender and the receiver to agree on the same secret key? What should happen if the attacker later forwards update information that was generated by the sender to the receiver?

In our security model we choose to insist on two straightforward policies in this scenario. The first is that whenever update information not generated by the sender is accepted by the receiver, even full knowledge of the key that the receiver has generated should not leak any information about other correctly generated keys. The second is that at any fixed point in time, if update information generated by the sender is accepted by the receiver then the receiver should agree with the sender on what the corresponding output key is, and the adversary should not be able to distinguish the shared output key from random.

Key indistinguishability of ratcheted key exchange schemes. Consider game KIND on the left side of Fig. 5 associated to a ratcheted key exchange scheme RKE and an adversary $\mathcal{D}$. The advantage of $\mathcal{D}$ in breaking the KIND security of RKE is defined as $\mathsf{Adv}^{\mathsf{kind}}_{\mathsf{RKE},\mathcal{D}} = 2\Pr[\mathrm{KIND}^{\mathcal{D}}_{\mathsf{RKE}}] - 1$.

The adversary is given the sender's static key stk_s as well as access to oracles RatSend, RatRec, Exp, ChSend, and ChRec. It can call oracle RatSend to receive update information upd from the sender, and it can call oracle RatRec to pass arbitrary update information to the receiver. Oracle Exp returns the current secrets sek_s, k_s possessed by the sender as well as the random seed r that was used to create the most recent upd in RatSend. Note that according to our notation convention from Section 2, integer variable r is assumed to be initialized to 0 at the beginning of the security game; this value will be returned if adversary calls Exp prior to RatSend.

The challenge oracles ChSend and ChRec provide the adversary with keys k_s and k_r in the

real world (when $b = 1$), or with uniformly random bit strings in the random world (when $b = 0$). The goal of the adversary is to distinguish between these two worlds. To disallow trivial attacks the game makes use of tables op and auth (initialized as empty) as well as a boolean flag restricted (initialized as false). Specifically, op keeps track of the oracle calls made by the adversary and is used to ensure that it can not trivially win the game by calling oracle Exp to get secrets that were used for one of the challenge queries. Table auth keeps track of the update information *upd* generated by RatSend so that we can set the flag restricted whenever the adversary has taken advantage of an Exp query to send maliciouly generated *upd* to RatRec. In this case we do not expect the receiver's key k_r to look random or match the sender's key k_s so ChRec is "restricted" and will return k_r in both the real and random worlds.

Authenticity of key exchange. Our security definition implicitly requires the authenticity of key exchange. Specifically, assume that an adversary can violate the authenticity in a non-trivial way, meaning without using Exp oracle to acquire the relevant secrets. This means that the adversary can construct malicious update information upd^* that is accepted by the receiver, while not setting the restricted flag to true. By making the receiver accept upd^*, the adversary achieves the situation when the sender and the receiver produce different output keys $k_s \neq k_r$. Now adversary can call oracles ChSend and ChRec to get both keys and compare them to win the game. In the real world ($b = 1$) the returned keys will be different, whereas in the random world ($b = 0$) they will be the same. We formalize this attack in Appendix B.

Allowing recovery from failures. Consider a situation when an attacker steals all sender's secrets, and hence has an ability to impersonate the sender. It can drop all further packets sent by the sender and instead use the exposed secrets to agree on its own shared secret keys with the receiver. In the security game this corresponds to the case when the adversary calls Exp and then starts calling oracle RatRec with maliciously generated update information *upd*. This sets the restricted flag to true, making the ChRec oracle always return the real receiver's key k_r regardless of the value of game's challenge bit b. The design decision at this point is – do we want to allow the game to recover from this state, meaning should the restricted flag be ever set back to false?

Our decision on this matter was determined by the two "policies" discussed above. As long as the adversary keeps sending maliciously generated update information *upd*, the restricted flag will remain true. In this case, the real receiver's key k_r returned from ChRec should be of no help in distinguishing the real sender's key k_s from random, as desired from the first policy. To match the second policy, the next time adversary forwards the *upd* generated by the sender (i.e. $upd = \mathsf{auth}[i_r]$) to RatRec, if *upd* is accepted by the receiver then the restricted flag is set back to false. This makes the output of ChRec again depend on the challenge bit, thus requiring k_r to be equal to k_s and indistinguishable from random.

Alternative treatment of restricted flag. Our security definition of KIND can be strengthened by making it never reset the restricted flag back to false. Instead, the game could require that if the adversary exposes sender's secrets and uses them to agree on its own shared output key with the receiver, then all the communication between the sender and the receiver should be disrupted. Meaning that any future attempt to simply forward sender's update information *upd* to the receiver should result in RatRec rejecting it. Otherwise adversary would be defined to win the game. This can be formalized in a number of ways. Our construction of ratcheted key exchange from Section 4.3 should be secure for a stronger definition like that, but would likely require stronger assumptions to prove.

Game $\mathrm{KIND}^{\mathcal{D}}_{\mathsf{RKE}}$	Game $\mathrm{RAE}^{\mathcal{A}}_{\mathsf{RE}}$
$b \leftarrow\$ \{0,1\}$; $i_s \leftarrow 0$; $i_r \leftarrow 0$ $(k, sek_s, (stk_s, stk_r, sek_r)) \leftarrow\$ \mathsf{RKE.IKg}$ $k_s \leftarrow k$; $k_r \leftarrow k$ $b' \leftarrow\$ \mathcal{D}^{\textsc{RatSend},\textsc{RatRec},\textsc{Exp},\textsc{ChSend},\textsc{ChRec}}(stk_s)$ Return $(b' = b)$ $\underline{\textsc{RatSend}}$ $r \leftarrow\$ \mathsf{RKE.RS}$ $(sek_s, k_s, upd) \leftarrow \mathsf{RKE.SKg}(stk_s, sek_s; r)$ $\mathsf{auth}[i_s] \leftarrow upd$; $i_s \leftarrow i_s + 1$ Return upd $\underline{\textsc{RatRec}(upd)}$ $z \leftarrow\$ \mathsf{RKE.RKg}(stk_s, stk_r, sek_r, upd, k_r)$ $(sek_r, k_r, acc) \leftarrow z$ If not acc then return false If $\mathsf{op}[i_r] =$ "exp" then $\mathsf{restricted} \leftarrow \mathsf{true}$ If $upd = \mathsf{auth}[i_r]$ then $\mathsf{restricted} \leftarrow \mathsf{false}$ $i_r \leftarrow i_r + 1$; Return true $\underline{\textsc{Exp}}$ If $\mathsf{op}[i_s] =$ "ch" then return $\perp$ $\mathsf{op}[i_s] \leftarrow$ "exp" ; Return (r, sek_s, k_s) $\underline{\textsc{ChSend}}$ If $\mathsf{op}[i_s] =$ "exp" then return $\perp$ $\mathsf{op}[i_s] \leftarrow$ "ch" If $\mathsf{rkey}[i_s] = \perp$ then $\mathsf{rkey}[i_s] \leftarrow\$ \{0,1\}^{\mathsf{RKE.kl}}$ If $b = 1$ then return k_s else return $\mathsf{rkey}[i_s]$ $\underline{\textsc{ChRec}}$ If $\mathsf{restricted}$ then return k_r If $\mathsf{op}[i_r] =$ "exp" then return $\perp$ $\mathsf{op}[i_r] \leftarrow$ "ch" If $\mathsf{rkey}[i_r] = \perp$ then $\mathsf{rkey}[i_r] \leftarrow\$ \{0,1\}^{\mathsf{RKE.kl}}$ If $b = 1$ then return k_r else return $\mathsf{rkey}[i_r]$	$b \leftarrow\$ \{0,1\}$; $i_s \leftarrow 0$; $i_r \leftarrow 0$ $(k, sek_s, (stk_s, stk_r, sek_r)) \leftarrow\$ \mathsf{RE.IKg}$ $k_s \leftarrow k$; $k_r \leftarrow k$ $b' \leftarrow\$ \mathcal{A}^{\textsc{RatSend},\textsc{RatRec},\textsc{Exp},\textsc{Enc},\textsc{Dec}}(stk_s)$ Return $(b' = b)$ $\underline{\textsc{RatSend}}$ $r \leftarrow\$ \mathsf{RE.RS}$ $(sek_s, k_s, upd) \leftarrow \mathsf{RE.SKg}(stk_s, sek_s; r)$ $\mathsf{auth}[i_s] \leftarrow upd$; $i_s \leftarrow i_s + 1$ Return upd $\underline{\textsc{RatRec}(upd)}$ $z \leftarrow\$ \mathsf{RE.RKg}(stk_s, stk_r, sek_r, upd, k_r)$ $(sek_r, k_r, acc) \leftarrow z$ If not acc then return false If $\mathsf{op}[i_r] =$ "exp" then $\mathsf{restricted} \leftarrow \mathsf{true}$ If $upd = \mathsf{auth}[i_r]$ then $\mathsf{restricted} \leftarrow \mathsf{false}$ $i_r \leftarrow i_r + 1$; Return true $\underline{\textsc{Exp}}$ If $\mathsf{op}[i_s] =$ "ch" then return $\perp$ $\mathsf{op}[i_s] \leftarrow$ "exp" ; Return (r, sek_s, k_s) $\underline{\textsc{Enc}(n, m, h)}$ If $\mathsf{op}[i_s] =$ "exp" then return $\perp$ $\mathsf{op}[i_s] \leftarrow$ "ch" If $(i_s, n) \in U$ then return $\perp$ $c_1 \leftarrow \mathsf{RE.Enc}(k_s, n, m, h)$ $c_0 \leftarrow\$ \{0,1\}^{\mathsf{RE.cl}(\lvert m \rvert)}$; $U \leftarrow U \cup \{(i_s, n)\}$ $S \leftarrow S \cup \{(i_s, n, c_b, h)\}$ Return c_b $\underline{\textsc{Dec}(n, c, h)}$ If $\mathsf{restricted}$ then Return $\mathsf{RE.Dec}(k_r, n, c, h)$ If $\mathsf{op}[i_r] =$ "exp" then return $\perp$ $\mathsf{op}[i_r] \leftarrow$ "ch" If $(i_r, n, c, h) \in S$ then return $\perp$ $m \leftarrow \mathsf{RE.Dec}(k_r, n, c, h)$ If $b = 1$ then return m else return $\perp$

Figure 5: Games defining key indistinguishability of ratcheted key exchange scheme RKE, and authenticated encryption security of ratcheted encryption scheme RE.

4.3 Construction of a ratcheted key exchange scheme

In this section we construct a ratcheted key exchange scheme, and discuss some design considerations by presenting a number of attacks that our scheme manages to evade. In Section 4.4 we will deduce a bound on the success of any adversary attacking the KIND security of our scheme. The idea of our construction is as follows. We let the sender and the receiver perform the Diffie-Hellman key exchange. The receiver's static key contains a secret DH exponent $stk_r = y$ and the sender's static key contains the corresponding public value $stk_s = g^y$ (working in some cyclic group

Algorithm $\mathsf{RKE.IKg}$

$k \leftarrow\$ \{0,1\}^{\mathsf{RKE.kl}}$
$fk \leftarrow\$ \{0,1\}^{\mathsf{F.kl}}$
$hk \leftarrow\$ \{0,1\}^{\mathsf{H.kl}}$
$g \leftarrow\$ \mathbb{G}^* \; ; \; y \leftarrow\$ \mathbb{Z}_p$
$stk_s \leftarrow (hk, g, g^y) \; ; \; stk_r \leftarrow y$
$sek_s \leftarrow (0, fk)$
$sek_r \leftarrow (0, fk)$
$z \leftarrow (k, sek_s, (stk_s, stk_r, sek_r))$
Return z

Algorithm $\mathsf{RKE.SKg}((hk, g, Y), (i_s, fk_s); r)$

$x \leftarrow r \; ; \; X \leftarrow g^x \; ; \; \sigma \leftarrow \mathsf{F.Ev}(fk_s, X)$
$s \leftarrow \mathsf{H.Ev}(hk, i_s \,\|\, \sigma \,\|\, X \,\|\, Y^x) \; ; \; k_s \leftarrow s[1 \ldots \mathsf{RKE.kl}]$
$fk_s \leftarrow s[\mathsf{RKE.kl}+1 \ldots \mathsf{RKE.kl}+\mathsf{F.kl}]$
Return $((i_s+1, fk_s), k_s, (X, \sigma))$

Algorithm $\mathsf{RKE.RKg}((hk, g, Y), y, (i_r, fk_r), (X, \sigma), k_r)$

$acc \leftarrow (\sigma = \mathsf{F.Ev}(fk_r, X))$
If not acc then return $((i_r, fk_r), k_r, acc)$
$s \leftarrow \mathsf{H.Ev}(hk, i_r \,\|\, \sigma \,\|\, X \,\|\, X^y) \; ; \; k_r \leftarrow s[1 \ldots \mathsf{RKE.kl}]$
$fk_r \leftarrow s[\mathsf{RKE.kl}+1 \ldots \mathsf{RKE.kl}+\mathsf{F.kl}]$
Return $((i_r+1, fk_r), k_r, acc)$

Figure 6: Ratcheted key exchange scheme $\mathsf{RKE} = \mathsf{RATCHET\text{-}KE}[\mathbb{G}, \mathsf{F}, \mathsf{H}]$.

Adversary $\mathcal{D}_1(stk_s)$

$(hk, g, Y) \leftarrow stk_s$
$x \leftarrow\$ \mathbb{Z}_p \; ; \; \textsc{RatRec}(g^x)$
$k_r \leftarrow \textsc{ChRec}$
$k'_r \leftarrow \mathsf{H.Ev}(hk, Y^x)$
If $k'_r = k_r$ then return 1
Else return 0

Adversary $\mathcal{D}_2(stk_s)$

$(hk, g, Y) \leftarrow stk_s$
$x \leftarrow\$ \mathbb{Z}_p \; ; \; \textsc{RatRec}(g^x)$
$k_r \leftarrow \textsc{ChRec}$
$(r, fk_s, k_s) \leftarrow \textsc{Exp}$
$k \,\|\, fk \leftarrow \mathsf{H.Ev}(hk, fk_s \,\|\, Y^x)$
If $k = k_r$ then return 1
Else return 0

Adversary $\mathcal{D}_3(stk_s)$

$upd_0 \leftarrow \textsc{RatSend} \; ; \; \textsc{RatRec}(upd_0) \; ; \; k_s \leftarrow \textsc{ChSend}$
$upd_1 \leftarrow \textsc{RatSend} \; ; \; \textsc{RatRec}(upd_1)$
$(r, fk_s, k_s) \leftarrow \textsc{Exp} \; ; \; (X_0, \sigma_0) \leftarrow upd_0$
$\sigma \leftarrow \mathsf{F.Ev}(fk_s, X_0) \; ; \; upd_2 \leftarrow (X_0, \sigma)$
$\textsc{RatRec}(upd_2) \; ; \; k_r \leftarrow \textsc{ChRec}$
If $k_s = k_r$ then return 1 else return 0

Adversary $\mathcal{D}_4(stk_s)$

$(r, sek_s, k_s) \leftarrow \textsc{Exp}$
$((i_s^*, fk_s^*), k_s^*, upd^*) \leftarrow\$ \mathsf{RKE.SKg}(stk_s, sek_s)$
$upd_0 \leftarrow \textsc{RatSend} \; ; \; \textsc{RatRec}(upd^*)$
$upd_1 \leftarrow \textsc{RatSend} \; ; \; (X, \sigma) \leftarrow upd_1$
$\sigma^* \leftarrow \mathsf{F.Ev}(fk_s^*, X) \; ; \; upd^* \leftarrow (X, \sigma^*) \; ; \; \textsc{RatRec}(upd^*)$
$k_s \leftarrow \textsc{ChSend} \; ; \; k_r \leftarrow \textsc{ChRec}$
If $k_s = k_r$ then return 1 else return 0

Figure 7: Attacks against insecure variants of $\mathsf{RKE} = \mathsf{RATCHET\text{-}KE}[\mathbb{G}, \mathsf{F}, \mathsf{H}]$.

with generator g). In order to generate a new shared secret key, the sender picks its own secret exponent x and computes the output key (roughly) as $k_s = H(stk_s^x) = H(g^{xy})$, where H is some hash function. The sender then sends update information containing g^x to the receiver, enabling the latter to compute the same output key. In order to ensure the security of the key exchange, both parties use a shared MAC key, meaning the update information also includes a tag of g^x.

Note that the used MAC key should be regularly renewed in order to ensure that the scheme provides backward security against exposures. As a result, the output of applying the hash function on g^{xy} is also used to derive a new MAC key. The initial key generation provides both parties with a shared MAC key and a shared secret key that are sampled uniformly at random. The formal definition of our key exchange scheme is as follows.

Ratcheted key exchange scheme $\mathsf{RATCHET\text{-}KE}$. Let $\mathbb{G}$ be a cyclic group of order $p \in \mathbb{N}$, and let $\mathbb{G}^*$ denote the set of its generators. Let F be a function family such that $\mathsf{F.In} = \mathbb{G}$. Let H be a function family such that $\mathsf{H.In} = \{0,1\}^*$ and $\mathsf{H.ol} > \mathsf{F.kl}$. We build a ratcheted key exchange scheme $\mathsf{RKE} = \mathsf{RATCHET\text{-}KE}[\mathbb{G}, \mathsf{F}, \mathsf{H}]$ as defined in Fig. 6, with $\mathsf{RKE.kl} = \mathsf{H.ol} - \mathsf{F.kl}$ and $\mathsf{RKE.RS} = \mathbb{Z}_p$.

Design considerations. We will examine some of the design decisions of RKE by considering several

ratcheted key exchange schemes that are weakened versions of RKE, and corresponding adversaries that are able to successfully attack these schemes. The first two will omit the use of a MAC and thus be vulnerable to attacks where the adversary sends its own update information to RatRec without having called Exp first (though the second will have to make an expose query afterwards). In the latter two examples we consider variations of RKE that use fewer inputs to the hash function. Our adversaries against these schemes thereby justify the choices we made for the input to the hash function. For the sake of compactness we omit showing that the constructed KIND adversaries have access to oracles RatSend, RatRec, Exp, ChSend, ChRec, and we omit showing that oracle calls return any output whenever this output is not used by the adversary.

Schemes without a MAC. First let us consider changing RKE to not use its MAC F and instead simply use an unauthenticated g^x as its update information. For simplicity we will additionally assume that the only input to H is a group element g^{xy}. Consider adversary $\mathcal{D}_1$ shown in Fig. 7. It makes a RatRec query with a g^x of its own choice, then calls oracle ChRec and checks whether the key it received was real or random by comparing it to $\mathsf{H}(hk, Y^x)$. Referring to this weakened scheme as RKE_1, it is clear that $\mathsf{Adv}^{\mathsf{kind}}_{\mathsf{RKE}_1,\mathcal{D}_1} = 1 - 2^{-\mathsf{RKE.kl}}$.

Besides using a MAC, another way to prevent the specific attack given above would be to put a shared secret key *fk* into the hash function along with g^{xy} for every update. Let RKE_2 denote a version of RKE that still does not use a MAC but updates its keys with the hash function via $k \,\|\, fk \leftarrow \mathsf{H.Ev}(hk, fk \,\|\, g^{xy})$. An adversary like $\mathcal{D}_1$ will not work against RKE_2 because computing the new value of k requires knowing the secret value *fk*. But there is still a simple attack against RKE_2. Consider adversary $\mathcal{D}_2$ shown in Fig. 7. It works in the same way as $\mathcal{D}_1$ except it needs to make an expose query to obtain fk_s before it can compute k using the hash function. One subtle point to notice is that it is important that $\mathcal{D}_2$ calls Exp *after* its call to RatRec. Otherwise the restricted flag in KIND would have been set to true and ChRec would always return the real key (instead of returning a randomly chosen key when the challenge bit in KIND is set to 0). Having noticed this it is clear that $\mathsf{Adv}^{\mathsf{kind}}_{\mathsf{RKE}_2,\mathcal{D}_2} = 1 - 2^{-\mathsf{RKE.kl}}$.

In Appendix B we give an attack against *any* ratcheted encryption scheme, showing that if it is possible for an adversary to generate its own *upd* that the receiver will accept, than the adversary can use this ability to successfully attack the ratcheted encryption scheme. This proves that some sort of authentication is required for the update information if we want a scheme to be secure.

Authenticating the update information in the Double Ratchet algorithm. The default version of the Double Ratchet algorithm [19, 15] — which is used in the Signal protocol [21] — does not authenticate the update information. A single, one-sided version of this algorithm would evolve its keys in a way that is vaguely similar to the RKE_2 scheme discussed above, so it would not meet our security definition. This does not immediately lead to any real-world attacks, and could mean that our security definition is stronger than necessary. Furthermore, [15] describes the header encryption variant of the Double Ratchet algorithm. A single, one-sided version of this algorithm provides some form of authentication for update information and might meet our security definition.

Necessity of inputs to H. In the construction of RATCHET-KE, function $\mathsf{H}(hk, \cdot)$ takes a string $w = i \,\|\, \sigma_i \,\|\, g^{x_i} \,\|\, g^{x_i y}$ as input. The most straightforward part of w is $g^{x_i y}$, which provides unpredictability to ensure that the generated keys are indistinguishable from uniformly random strings. String w also includes the counter i, and the corresponding update information $upd_i = (g^{x_i}, \sigma_i)$. The inclusion of counter i in w ensures that an attacker cannot perform a "key-reuse" attack to make the receiver generate an output key that was already used before; we provide an example

of such attack below. We also describe a "key-collision" attack against the KIND security of the scheme that is prevented by including upd_i in w. Finally, note that our concatenation operator $\|$ is defined to produce uniquely decodable strings, so the mapping of $(i, \sigma_i, g^{x_i}, g^{x_i y})$ into string w is injective; this helps to avoid attacks that take advantage of malleable encodings.

Key-reuse attack. Game KIND makes sure that if challenge keys are acquired from the sender and the receiver for the same value of i (i.e. $i_s = i_r$), then these keys are consistent even if they are picked randomly. Otherwise it would be trivial to attack any ratcheted key exchange scheme. However, the game does not maintain such consistency between different values of i. Let RKE_3 denote RKE if it was changed to use only g^{xy} as input to the hash function. Consider the "key-reuse" attack $\mathcal{D}_3$ shown in Fig. 7 that exploits the above as follows. Adversary $\mathcal{D}_3$ starts by calling RatSend, RatRec and ChSend to get a sender's challenge key k_s. Note that if the challenge bit is $b = 1$ in game KIND, then k_s equals to $\mathsf{H.Ev}(hk, Y^x)$ for some exponent x generated during RatSend. Next, the adversary calls both RatSend and RatRec to ratchet the key forward, in order to be able to make Exp queries. It calls Exp to get fk_s so that it can re-authenticate the same value of $X = g^x$ that was used for the sender's challenge query. Then it sends X and its new MAC tag σ to the receiver, which sets the restricted flag true. The latter means that calling ChRec results in getting the receiver's real output key regardless of the challenge bit. If this key is equal to the previously learned sender's challenge key then it is highly likely that the challenge bit b equals 1, otherwise it must be 0. This gives the advantage of $\mathsf{Adv}^{\mathsf{kind}}_{\mathsf{RKE}_3,\mathcal{D}_3} = 1 - 2^{-\mathsf{RKE.kl}}$.

Key-collision attacks. We now describe the final attack idea that does not work against our construction but would have been possible if the update information $upd = (g^{x_i}, \sigma)$ was not included in the hash function. Consider changing $\mathsf{RATCHET\text{-}KE}[\mathbb{G}, \mathsf{F}, \mathsf{H}]$ to have $\mathsf{H}(hk, \cdot)$ take inputs of the form $w = i \,\|\, g^{x_i y}$. Call this scheme RKE_4. This enables the following attack, as defined by the adversary $\mathcal{D}_4$ in Fig. 7. Assume that an attacker compromises the sender's keys k_s and fk_s and immediately uses the compromised authenticity to establish new keys k_s^* and fk_s^*, shared between the attacker and the receiver. Now let $upd = (X, \sigma)$ be the next update information produced by the sender. The attacker can construct malicious update information $upd^* = (X, \sigma^*)$, where $\sigma^* = \mathsf{F.Ev}(fk_s^*, X)$, and send it to the receiver. The receiver would accept upd^* and use the output of $\mathsf{H.Ev}(hk, i \,\|\, X^y)$ as new key material, resulting in the same keys as those generated by the sender. Now the the receiver and the sender share an output key, while the restricted flag is set true, so checking whether the output of the two challenge oracles is the same yields a good attack.

We will not give the exact advantage of $\mathcal{D}_4$. If σ^* and σ happen to be exactly the same, then the restricted flag would be set back to false and the attack would fail because the two keys received from the sender's and the reciever's challenge oracles would be the same regardless of game's challenge bit. But if $\sigma^* = \sigma$ was likely to occur then the ratcheted key exchange scheme would be insecure for other reasons. One could formalize this by building a second adversary against RKE_4 to show that one of the two adversaries must have a high advantage. For the purpose of this section we simply note that this event is extremely unlikely to occur for any typical choice of hash function and MAC.

4.4 Security proof for our ratcheted key exchange scheme

In previous section we showed that several variations of our ratcheted key exchange scheme $\mathsf{RKE} = \mathsf{RATCHET\text{-}KE}[\mathbb{G}, \mathsf{F}, \mathsf{H}]$ are insecure. In this section we will prove that our scheme is secure. We now present our theorem bounding the advantage of an adversary breaking the KIND-security of

RKE to the SUFCMA-security of F and to the ODHE-security of $\mathbb{G}, \mathsf{H}$.

Theorem 4.1 *Let $\mathbb{G}$ be a cyclic group of order $p \in \mathbb{N}$, and let $\mathbb{G}^*$ denote the set of its generators. Let F be a function family such that $\mathsf{F.In} = \mathbb{G}$. Let H be a function family such that $\mathsf{H.In} = \{0,1\}^*$ and $\mathsf{H.ol} > \mathsf{F.kl}$. Let $\mathsf{RKE} = \mathsf{RATCHET\text{-}KE}[\mathbb{G}, \mathsf{F}, \mathsf{H}]$. Let $\mathcal{D}$ be an adversary attacking the* KIND-*security of RKE that makes q_{RatSend} queries to its* RatSend *oracle, q_{RatRec} queries to its* RatRec *oracle, q_{Exp} queries to its* Exp *oracle, q_{ChSend} queries to its* ChSend *oracle, and q_{ChRec} queries to its* ChRec *oracle. Then there is an adversary $\mathcal{F}$ attacking the* SUFCMA-*security of F, and adversaries $\mathcal{O}_1, \mathcal{O}_2$ attacking the* ODHE-*security of $\mathbb{G}, \mathsf{H}$, such that*

$$\mathsf{Adv}^{\mathsf{kind}}_{\mathsf{RKE},\mathcal{D}} \leq 2\cdot(q_{\text{RatSend}}+1) \cdot \mathsf{Adv}^{\mathsf{sufcma}}_{\mathsf{F},\mathcal{F}} + 2\cdot q_{\text{RatSend}}\cdot \mathsf{Adv}^{\mathsf{odhe}}_{\mathbb{G},\mathsf{H},\mathcal{O}_1} + 2\cdot \mathsf{Adv}^{\mathsf{odhe}}_{\mathbb{G},\mathsf{H},\mathcal{O}_2}.$$

Adversary $\mathcal{F}$ makes at most q_{RatSend} queries to its Tag *oracle and q_{RatRec} queries to its* Verify *oracle. Adversary $\mathcal{O}_1$ makes at most q_{RatSend} queries to its* Up *oracle, 2 queries to its* Ch *oracle, q_{Exp} queries to its* Exp *oracle, and $q_{\text{RatSend}}+q_{\text{RatRec}}-2$ queries to its* Hash *oracle. Adversary $\mathcal{O}_2$ makes at most q_{RatSend} queries to its* Up *oracle, $q_{\text{RatSend}} + q_{\text{RatRec}}$ queries to its* Ch *oracle,q_{Exp} queries to its* Exp *oracle, and $q_{\text{RatRec}} + q_{\text{Exp}}$ queries to its* Hash *oracle. Each of $\mathcal{F}$, $\mathcal{O}_1$, $\mathcal{O}_2$ has a running time approximately that of $\mathcal{D}$.*

The proof requires careful attention to detail due to subtleties. The most natural proof method may be to proceed one RatSend query at a time, first replacing the output of the hash function with random bits (unless an expose happens) and then using the security of the MAC to argue that the adversary cannot produce any modified update information that will be accepted by the receiver without exposing. But there is a subtle flaw with this proof technique. The adversary may attempt to create a forged *upd* before it has decided whether to expose. In this case we need to check the validity of their forgery with a MAC key, before we know whether it should be random or a valid output of the hash function.

To avoid this problem we first use a hybrid argument to show that no such forgery is possible before replacing all non-exposed keys with random. We proceed one RatSend query at a time, showing that we can temporarily replace the key with random when checking the sort of attempted forgery described above. This then allows us to use the security of the MAC to assume that the forgery attempt failed without us having to commit to a key to verify with. We thus are able to show one step at a time that all such forgery attempts can be assumed to fail without having to check.

Once this is done, we are never forced to use a key before the adversary has committed to whether it will perform a relevant exposure of the secret state. As such we can safely delay our decision of whether or not the key should be replaced by random values until it is known whether an expose will happen. This allows us to use the ODHE security of H and $\mathbb{G}$ to argue that we can replace all of the generated keys with randomness, only using H to generate the real keys at the last moment whenever an expose query is made.

Proof of Theorem 4.1: Consider the sequence of games shown in Fig. 8. Lines not annotated with comments are common to all games. $\mathrm{G}_{0,0}$ is identical to $\mathrm{KIND}^{\mathcal{D}}_{\mathsf{RKE}}$ with the code of RKE inserted. Additionally, a flag $\mathsf{unchanged}$ has been added. This flag keeps track of whether the most recent update information was passed unchanged from the sender to the receiver and thus the keys k_r and fk_r should be indistinguishable from random to adversary $\mathcal{D}$. In this case, the adversary should not be able to create update information *upd* that is accepted by RatRec unless it calls Exp or forwards along the *upd* generated by the sender. We prove this with a hybrid

argument over the games $\mathrm{G}_{0,0}, \ldots, \mathrm{G}_{0,q_{\textsc{RatSend}}+1}$. Game $\mathrm{G}_{0,j}$ assumes forgery attempts fail for the first j keys, sets a bad flag if $\mathcal{D}$ is successful at forging against the $(j+1)$-th key, and performs normally for all following keys. Game $\mathrm{G}^*_{0,j}$ is the same except it also acts as if $\mathcal{D}$ failed to forge even when the bad flag is set. Thus, from the perpective of an adversary $\mathrm{G}^*_{0,j}$ is simply assumping that forgery attempts fail for the first $j+1$ keys, making it equivalent to $\mathrm{G}_{0,j+1}$. Thus for all $j \in \{0, \ldots, q_{\textsc{RatSend}}\}$,

$$\Pr[\mathrm{G}_{0,0}] = \Pr[\mathrm{KIND}^{\mathcal{D}}_{\mathsf{RKE}}] \qquad \text{and} \qquad \Pr[\mathrm{G}^*_{0,j}] = \Pr[\mathrm{G}_{0,j+1}].$$

Furthermore, for all $j \in \{1, \ldots, q_{\textsc{RatSend}}\}$, games $\mathrm{G}_{0,j}$ and $\mathrm{G}^*_{0,j}$ are identical until bad, so the fundamental lemma of game playing [6] gives:

$$\Pr[\mathrm{G}_{0,j}] - \Pr[\mathrm{G}^*_{0,j}] \leq \Pr[\mathsf{bad}^{\mathrm{G}^*_{0,j}}],$$

where $\Pr[\mathsf{bad}^{\mathsf{Q}}]$ denotes the probability of setting the bad flag in game Q.

We cannot directly bound $\Pr[\mathsf{bad}^{\mathrm{G}^*_{0,j}}]$ using the security of F because the key being used for F is chosen as output from H instead of uniformly at random, consider the relationship between games $\mathrm{G}^*_{0,j}$ and I_j (the latter also shown in Fig. 8). Game I_j is identical to $\mathrm{G}^*_{0,j}$, except that in I_j the output of hash function H is replaced with a uniformly random string whenever $i+1=j$ (thus the key used to check whether bad should be set when $i=j$ is uniformly random).

Note that when $j=0$ the games $\mathrm{G}^*_{0,0}$ and I_0 are identical so $\Pr[\mathsf{bad}^{\mathrm{G}^*_{0,0}}] = \Pr[\mathsf{bad}^{\mathrm{I}_0}]$. For other values of j we relate the probability that these games set bad to the advantage of the oracle Diffie-Hellman adversary $\mathcal{O}_1$ that is defined in Fig. 10. Adversary $\mathcal{O}_1$ picks j' at random and then uses its oracles to simulate $\mathrm{G}^*_{0,j}$ or I_j. Then if the bad flag is set it sets a bit b' equal to 1. This bit is ultimately returned by $\mathcal{O}$. Thus the probability that $\mathcal{O}$ outputs 1 is exactly the probability that the bad flag would be set in the game it is simulating.

Let b_{odhe} denote the challenge bit in game $\mathrm{ODHE}^{\mathcal{O}_1}_{\mathbb{G},\mathsf{H}}$, and let b' denote the corresponding guess made by the adversary $\mathcal{O}_1$. Let j' be the value sampled in the first step of $\mathcal{O}_1$. For each choice of j', adversary $\mathcal{O}_1$ perfectly simulates the view of $\mathcal{D}$ in either $\mathrm{G}^*_{0,j'}$ or $\mathrm{I}_{j'}$ depending on whether its $\textsc{Ch}$ oracle is returning real output of the hash function or a random value. If $\mathcal{D}$ performs an action that would prevent bad from being set (such as calling $\textsc{Exp}$ when $i_s = j'$) then $\mathcal{O}_1$ no longer perfectly simulates the view of $\mathcal{D}$, but it does not matter for our analysis because we already know bad (and thus b') will not be set. So for all $j \in \{1, \ldots, q_{\textsc{RatSend}}\}$, we have

$$\Pr[\mathsf{bad}^{\mathrm{G}^*_{0,j}}] = \Pr[\, b' = 1 \,|\, b_{\mathsf{odhe}} = 1, j' = j \,],$$
$$\Pr[\mathsf{bad}^{\mathrm{I}_j}] = \Pr[\, b' = 1 \,|\, b_{\mathsf{odhe}} = 0, j' = j \,].$$

Combining the above for all values of j (using $\Pr[\mathsf{bad}^{\mathrm{G}^*_{0,0}}] = \Pr[\mathsf{bad}^{\mathrm{G}i_s}]$) gives

$$\begin{aligned}
&\mathsf{Adv}^{\mathsf{odhe}}_{\mathbb{G},\mathsf{H},\mathcal{O}_1} = \Pr[\, b' = 1 \,|\, b_{\mathsf{odhe}} = 1 \,] - \Pr[\, b' = 1 \,|\, b_{\mathsf{odhe}} = 0 \,] \\
&= \sum_{j=1}^{q_{\textsc{RatSend}}} \Pr[j = j'](\Pr[\mathsf{bad}^{\mathrm{G}^*_{0,j}}] - \Pr[\mathsf{bad}^{\mathrm{I}_j}]) = \sum_{j=0}^{q_{\textsc{RatSend}}} \frac{\Pr[\mathsf{bad}^{\mathrm{G}^*_{0,j}}] - \Pr[\mathsf{bad}^{\mathrm{I}_j}]}{q_{\textsc{RatSend}}}.
\end{aligned}$$

Note that we were able to change the starting index of j for that last summation because $\Pr[\mathsf{bad}^{\mathrm{G}^*_{0,0}}] = \Pr[\mathsf{bad}^{\mathrm{I}_0}]$, as we noted before.

Games $\mathrm{G}_{0,j}$,$\mathrm{G}^*_{0,j}$,I_j

$b \leftarrow\$ \{0,1\}$; $i_s \leftarrow 0$; $i_r \leftarrow 0$; $\mathsf{unchanged} \leftarrow \mathsf{true}$; $rand \leftarrow\$ \{0,1\}^{\mathsf{H.ol}}$
$k_s \leftarrow\$ \{0,1\}^{\mathsf{RKE.kl}}$; $k_r \leftarrow k_s$; $fk_s \leftarrow\$ \{0,1\}^{\mathsf{F.kl}}$; $fk_r \leftarrow fk_s$
$hk \leftarrow\$ \{0,1\}^{\mathsf{H.kl}}$; $g \leftarrow\$ \mathbb{G}^*$; $y \leftarrow\$ \mathbb{Z}_p$; $stk_s \leftarrow (hk, g, g^y)$
$b' \leftarrow\$ \mathcal{D}^{\textsc{RatSend},\textsc{RatRec},\textsc{Exp},\textsc{ChSend},\textsc{ChRec}}(stk_s)$; Return $(b' = b)$

RatSend
If $\mathsf{op}[i_s] = \bot$ then $\mathsf{op}[i_s] \leftarrow$ "ch"
$x \leftarrow\$ \mathbb{Z}_p$; $\sigma \leftarrow \mathsf{F.Ev}(fk_s, g^x)$; $upd \leftarrow (g^x, \sigma)$
$s \leftarrow \mathsf{H.Ev}(hk, i_s \parallel \sigma \parallel g^x \parallel g^{xy})$
If $i_s + 1 = j$ then $s \leftarrow rand$ // I_j
$\mathsf{auth}[i_s] \leftarrow upd$; $i_s \leftarrow i_s + 1$; $k_s \leftarrow s[1 \ldots \mathsf{RKE.kl}]$
$fk_s \leftarrow s[\mathsf{RKE.kl} + 1 \ldots \mathsf{RKE.kl} + \mathsf{F.kl}]$; Return upd

RatRec(upd)
$(X, \sigma) \leftarrow upd$
If $\mathsf{unchanged}$ and $(\mathsf{op}[i_r] \neq$ "exp") and $(upd \neq \mathsf{auth}[i_r])$ then
  If $i_r < j$ then return false
  If $i_r = j$ then
    If $\sigma \neq \mathsf{F.Ev}(fk_r, X)$ then return false
    $\mathsf{bad} \leftarrow \mathsf{true}$
    Return false // $\mathrm{G}^*_{0,j}$,I_j
If $\sigma \neq \mathsf{F.Ev}(fk_r, X)$ then return false
If $\mathsf{op}[i_r] =$ "exp" then $\mathsf{restricted} \leftarrow \mathsf{true}$
If $upd = \mathsf{auth}[i_r]$ then
  $\mathsf{unchanged} \leftarrow \mathsf{true}$; $\mathsf{restricted} \leftarrow \mathsf{false}$
Else
  $\mathsf{unchanged} \leftarrow \mathsf{false}$
$s \leftarrow \mathsf{H.Ev}(hk, i_r \parallel \sigma \parallel X \parallel X^y)$
If $i_r + 1 = j$ then $s \leftarrow rand$ // I_j
$i_r \leftarrow i_r + 1$; $k_r \leftarrow s[1 \ldots \mathsf{RKE.kl}]$
$fk_r \leftarrow s[\mathsf{RKE.kl} + 1 \ldots \mathsf{RKE.kl} + \mathsf{F.kl}]$; Return true

Exp
If $\mathsf{op}[i_s] =$ "ch" then return $\bot$
$\mathsf{op}[i_s] \leftarrow$ "exp" ; Return $(x, (i_s, fk_s), k_s)$

ChSend
If $\mathsf{op}[i_s] =$ "exp" then return $\bot$
$\mathsf{op}[i_s] \leftarrow$ "ch"
If $\mathsf{rkey}[i_s] = \bot$ then $\mathsf{rkey}[i_s] \leftarrow\$ \{0,1\}^{\mathsf{RKE.kl}}$
If $b = 1$ then return k_s else return $\mathsf{rkey}[i_s]$

ChRec
If $\mathsf{restricted}$ then return k_r
If $\mathsf{op}[i_r] =$ "exp" then return $\bot$
$\mathsf{op}[i_r] \leftarrow$ "ch"
If $\mathsf{rkey}[i_r] = \bot$ then $\mathsf{rkey}[i_r] \leftarrow\$ \{0,1\}^{\mathsf{RKE.kl}}$
If $b = 1$ then return k_r else return $\mathsf{rkey}[i_r]$

Figure 8: Games $\mathrm{G}_{0,j}, \mathrm{G}^*_{0,j}, \mathrm{I}_j$ for proof of Theorem 4.1.

To complete the hybrid argument part of the proof, we can finally bound the probability that bad gets set true in I_j. Doing so requires adversary $\mathcal{D}$ to successfully forge a MAC tag for a uni-

Games G_1–G_2

$b \leftarrow\$ \{0,1\}$; $i_s \leftarrow 0$; $i_r \leftarrow 0$; $\mathsf{unchanged} \leftarrow \mathsf{true}$
$k_s[0] \leftarrow\$ \{0,1\}^{\mathsf{RKE.kl}}$; $k_r \leftarrow k_s[0]$; $fk_s[0] \leftarrow\$ \{0,1\}^{\mathsf{F.kl}}$; $fk_r \leftarrow fk_s[0]$
$hk \leftarrow\$ \{0,1\}^{\mathsf{H.kl}}$; $g \leftarrow\$ \mathbb{G}^*$; $y \leftarrow\$ \mathbb{Z}_p$; $stk_s \leftarrow (hk, g, g^y)$
$b' \leftarrow\$ \mathcal{D}^{\textsc{RatSend},\textsc{RatRec},\textsc{Exp},\textsc{ChSend},\textsc{ChRec}}(stk_s)$; Return $(b' = b)$

RatSend

If $\mathsf{op}[i_s] = \perp$ then $\mathsf{op}[i_s] \leftarrow$ "ch"
$x \leftarrow\$ \mathbb{Z}_p$; $\sigma \leftarrow \mathsf{F.Ev}(fk_s[i_s], g^x)$; $upd \leftarrow (g^x, \sigma)$
$s \leftarrow \mathsf{H.Ev}(hk, i_s \parallel \sigma \parallel g^x \parallel g^{xy})$ // G_1
$s \leftarrow\$ \{0,1\}^{\mathsf{H.ol}}$ // G_2
$\mathsf{auth}[i_s] \leftarrow upd$; $i_s \leftarrow i_s + 1$; $k_s[i_s] \leftarrow s[1 \ldots \mathsf{RKE.kl}]$
$fk_s[i_s] \leftarrow s[\mathsf{RKE.kl} + 1 \ldots \mathsf{RKE.kl} + \mathsf{F.kl}]$; Return upd

RatRec(upd)

$(X, \sigma) \leftarrow upd$
If $\mathsf{unchanged}$ and $(\mathsf{op}[i_r] \neq$ "exp") and $(upd \neq \mathsf{auth}[i_r])$ then
  Return false
If $\mathsf{unchanged}$ then $fk_r \leftarrow fk_s[i_r]$
If $(\sigma \neq \mathsf{F.Ev}(fk_r, X))$ then return false
If $\mathsf{op}[i_r] =$ "exp" then $\mathsf{restricted} \leftarrow \mathsf{true}$
If $upd = \mathsf{auth}[i_r]$
  $\mathsf{unchanged} \leftarrow \mathsf{true}$; $\mathsf{restricted} \leftarrow \mathsf{false}$; $i_r \leftarrow i_r + 1$
Else
  $\mathsf{unchanged} \leftarrow \mathsf{false}$
  $s \leftarrow \mathsf{H.Ev}(hk, i_r \parallel \sigma \parallel X \parallel X^y)$
  $i_r \leftarrow i_r + 1$; $k_r \leftarrow s[1 \ldots \mathsf{RKE.kl}]$
  $fk_r \leftarrow s[\mathsf{RKE.kl} + 1 \ldots \mathsf{RKE.kl} + \mathsf{F.kl}]$
Return true

Exp

If $\mathsf{op}[i_s] =$ "ch" then return $\perp$
$\mathsf{op}[i_s] \leftarrow$ "exp" ; $(X, \sigma) \leftarrow \mathsf{auth}[i_s - 1]$
$s \leftarrow \mathsf{H.Ev}(hk, (i_s - 1) \parallel \sigma \parallel X \parallel X^y)$; $k_s[i_s] \leftarrow s[1 \ldots \mathsf{RKE.kl}]$
$fk_s[i_s] \leftarrow s[\mathsf{RKE.kl} + 1 \ldots \mathsf{RKE.kl} + \mathsf{F.kl}]$
Return $(x, (i_s, fk_s[i_s]), k_s[i_s])$

ChSend

If $\mathsf{op}[i_s] =$ "exp" then return $\perp$
$\mathsf{op}[i_s] \leftarrow$ "ch"
If $\mathsf{rkey}[i_s] = \perp$ then $\mathsf{rkey}[i_s] \leftarrow\$ \{0,1\}^{\mathsf{RKE.kl}}$
If $b = 1$ then return $k_s[i_s]$ else return $\mathsf{rkey}[i_s]$

ChRec

If $\mathsf{restricted}$ then return k_r
If $\mathsf{op}[i_r] =$ "exp" then return $\perp$
$\mathsf{op}[i_r] \leftarrow$ "ch"
If $\mathsf{rkey}[i_r] = \perp$ then $\mathsf{rkey}[i_r] \leftarrow\$ \{0,1\}^{\mathsf{RKE.kl}}$
If $\mathsf{unchanged}$ then $k_r \leftarrow k_s[i_r]$
If $b = 1$ then return k_r else return $\mathsf{rkey}[i_r]$

Figure 9: Games G_1, G_2 for proof of Theorem 4.1.

formly random key, allowing us to reduce to the security of F. Formally, we use $\mathcal{D}$ to construct

Adversary $\mathcal{O}_1^{\text{Up},\text{Ch},\text{Exp},\text{Hash}}(hk, g, Y)$

$j' \leftarrow\$ \{1, \ldots, q_{\text{RatSend}}\}$; $b \leftarrow\$ \{0,1\}$; $b' \leftarrow 0$
$i_s \leftarrow 0$; $i_r \leftarrow 0$; $\mathsf{unchanged} \leftarrow \mathsf{true}$
$k_s \leftarrow\$ \{0,1\}^{\mathsf{RKE.kl}}$; $k_r \leftarrow k_s$
$fk_s \leftarrow\$ \{0,1\}^{\mathsf{F.kl}}$; $fk_r \leftarrow fk_s$; $stk_s \leftarrow (hk, g, Y)$
$\mathcal{D}^{\text{RatSendSim},\text{RatRecSim},\text{ExpSim},\text{ChSendSim},\text{ChRecSim}}(stk_s)$
Return b'

RatRecSim(upd)

$(X, \sigma) \leftarrow upd$
$\mathsf{forge} \leftarrow ((\mathsf{op}[i_r] \neq \text{“exp”}) \wedge (upd \neq \mathsf{auth}[i_r]))$
If $\mathsf{unchanged}$ and forge then
  If $i_r < j'$ then return false
  If $i_r = j'$ then
    If $\sigma \neq \mathsf{F.Ev}(fk_r, X)$ then return false
    $\mathsf{bad} \leftarrow \mathsf{true}$; $b' \leftarrow 1$; Return false
If $\sigma \neq \mathsf{F.Ev}(fk_r, X)$ then return false
If $\mathsf{op}[i_r] = \text{“exp”}$ then $\mathsf{restricted} \leftarrow \mathsf{true}$
If $upd = \mathsf{auth}[i_r]$ then
  $\mathsf{unchanged} \leftarrow \mathsf{true}$; $\mathsf{restricted} \leftarrow \mathsf{false}$
Else $\mathsf{unchanged} \leftarrow \mathsf{false}$
If $i_r + 1 \neq j'$ then $s \leftarrow \text{Hash}(i_r, \sigma \,\|\, X, X)$
Else $s \leftarrow \text{Ch}(\sigma \,\|\, X)$
$i_r \leftarrow i_r + 1$; $k_r \leftarrow s[1 \ldots \mathsf{RKE.kl}]$
$fk_r \leftarrow s[\mathsf{RKE.kl} + 1 \ldots \mathsf{RKE.kl} + \mathsf{F.kl}]$
Return true

ExpSim

If $\mathsf{op}[i_s] = \text{“ch”}$ then return $\perp$
$\mathsf{op}[i_s] \leftarrow \text{“exp”}$; $x \leftarrow \text{Exp}$
Return $(x, (i_s, fk_s), k_s)$

RatSendSim

If $\mathsf{op}[i_s] = \perp$ then $\mathsf{op}[i_s] \leftarrow \text{“ch”}$
$X \leftarrow \text{Up}$; $\sigma \leftarrow \mathsf{F.Ev}(fk_s, X)$
$upd \leftarrow (X, \sigma)$
If $i_s + 1 \neq j'$ then
  $s \leftarrow \text{Hash}(i_s, \sigma \,\|\, X, X)$
Else
  $s \leftarrow \text{Ch}(\sigma \,\|\, X)$
$\mathsf{auth}[i_s] \leftarrow upd$; $i_s \leftarrow i_s + 1$
$k_s \leftarrow s[1 \ldots \mathsf{RKE.kl}]$
$fk_s \leftarrow s[\mathsf{RKE.kl} + 1 \ldots \mathsf{RKE.kl} + \mathsf{F.kl}]$
Return upd

ChSendSim

If $\mathsf{op}[i_s] = \text{“exp”}$ then return $\perp$
$\mathsf{op}[i_s] \leftarrow \text{“ch”}$
If $\mathsf{rkey}[i_s] = \perp$ then
  $\mathsf{rkey}[i_s] \leftarrow\$ \{0,1\}^{\mathsf{RKE.kl}}$
If $b = 1$ then return k_s
Else return $\mathsf{rkey}[i_s]$

ChRecSim

If $\mathsf{restricted}$ then return k_r
If $\mathsf{op}[i_r] = \text{“exp”}$ then return $\perp$
$\mathsf{op}[i_r] \leftarrow \text{“ch”}$
If $\mathsf{rkey}[i_r] = \perp$ then
  $\mathsf{rkey}[i_r] \leftarrow\$ \{0,1\}^{\mathsf{RKE.kl}}$
If $b = 1$ then return k_r
Else return $\mathsf{rkey}[i_r]$

Figure 10: Adversary $\mathcal{O}_1$ for proof of Theorem 4.1.

an adversary $\mathcal{F}$ attacking the SUFCMA security of F. Adversary $\mathcal{F}$ (shown in Fig. 11) simulates adversary $\mathcal{D}$ and guesses when it will first create a forgery. $\mathcal{F}$ simulates game I_j for adversary $\mathcal{D}$ until that point, and uses its own SUFCMA oracles to answer $\mathcal{D}$'s queries at the time when it expects the forgery. Similar to the earlier case when $\mathcal{O}_1$ simulated $\mathcal{D}$, adversary $\mathcal{F}$ may fail to simulate I_j for adversary $\mathcal{D}$ when the latter performs certain actions that preclude bad from being set. This does not affect our analysis because we only require that if bad is set then $\mathcal{F}$ will return a successful forgery. Thus for $j \in \{0, \ldots, q_{\text{RatSend}}\}$, $\Pr[\mathsf{bad}^{\mathrm{I}_j}] \leq \Pr[\,\text{SUFCMA}_{\mathsf{F}}^{\mathcal{F}} \mid j' = j\,]$ which gives $\mathsf{Adv}_{\mathsf{F},\mathcal{F}}^{\mathsf{sufcma}} \geq (1/(q_{\text{RatRec}} + 1)) \sum_{j=0}^{q_{\text{RatRec}}} \Pr[\mathsf{bad}^{\mathrm{I}_j}]$.

The above work allows us to transition to game $\mathrm{G}_{0,q_{\text{RatSend}}+1}$ as shown in the following equations. From there we will move to games $\mathrm{G}_1, \mathrm{G}_2$ shown in Fig. 9. All of the summations below are from $j = 0$ to $j = q_{\text{RatSend}}$.

$$\Pr[\text{KIND}_{\mathsf{RKE}}^{\mathcal{D}}] = \Pr[\mathrm{G}_{0,0}] = \Pr[\mathrm{G}_{1,q_{\text{RatSend}}}] + \textstyle\sum_j \Pr[\mathrm{G}_{0,j}] - \Pr[\mathrm{G}_{0,j}^*]$$
$$\leq \Pr[\mathrm{G}_{1,q_{\text{RatSend}}}] + \textstyle\sum_j \Pr[\mathsf{bad}^{\mathrm{G}_{0,j}^*}]$$

Adversary $\mathcal{F}^{\textsc{Tag},\textsc{Verify}}$

$j' \leftarrow\$ \{0,\ldots,q_{\textsc{RatSend}}\}$; $b \leftarrow\$ \{0,1\}$
$i_s \leftarrow 0$; $i_r \leftarrow 0$; $\mathsf{unchanged} \leftarrow \mathsf{true}$
$rand \leftarrow\$ \{0,1\}^{\mathsf{H.ol}}$; $k_s \leftarrow\$ \{0,1\}^{\mathsf{RKE.kl}}$; $k_r \leftarrow k_s$
$fk_s \leftarrow\$ \{0,1\}^{\mathsf{F.kl}}$; $fk_r \leftarrow fk_s$; $hk \leftarrow\$ \{0,1\}^{\mathsf{H.kl}}$
$g \leftarrow\$ \mathbb{G}^*$; $y \leftarrow\$ \mathbb{Z}_p$; $stk_s \leftarrow (hk, g, g^y)$
$\mathcal{D}^{\textsc{RatSendSim},\textsc{RatRecSim},\textsc{ExpSim},\textsc{ChSendSim},\textsc{ChRecSim}}(stk_s)$

$\textsc{RatRecSim}(upd)$

$(X,\sigma) \leftarrow upd$
$\mathsf{forge} \leftarrow ((\mathsf{op}[i_r] \neq \text{“exp”}) \wedge (upd \neq \mathsf{auth}[i_r]))$
If $\mathsf{unchanged}$ and forge then
  If $i_r < j'$ then return false
  If $i_r = j'$ then
    If not $\textsc{Verify}(X,\sigma)$ then return false
    $\mathsf{bad} \leftarrow \mathsf{true}$
    Return false
If $(i_r = j')$ then
  If not $\textsc{Verify}(X,\sigma)$ then return false
Else
  If $\sigma \neq \mathsf{F.Ev}(fk_r, X)$ then return false
If $\mathsf{op}[i_r] =$ “exp” then $\mathsf{restricted} \leftarrow \mathsf{true}$
If $upd = \mathsf{auth}[i_r]$ then
  $\mathsf{unchanged} \leftarrow \mathsf{true}$; $\mathsf{restricted} \leftarrow \mathsf{false}$
Else
  $\mathsf{unchanged} \leftarrow \mathsf{false}$
$s \leftarrow \mathsf{H.Ev}(hk, i_r \,\|\, \sigma \,\|\, X \,\|\, X^y)$
If $i_r + 1 = j$ then $s \leftarrow rand$
$i_r \leftarrow i_r + 1$; $k_r \leftarrow s[1 \ldots \mathsf{RKE.kl}]$
$fk_r \leftarrow s[\mathsf{RKE.kl}+1 \ldots \mathsf{RKE.kl}+\mathsf{F.kl}]$
Return true

$\textsc{RatSendSim}$

If $\mathsf{op}[i_s] = \bot$ then $\mathsf{op}[i_s] \leftarrow$ “ch”
$x \leftarrow\$ \mathbb{Z}_p$
If $i_s = j'$ then $\sigma \leftarrow \textsc{Tag}(g^x)$
Else $\sigma \leftarrow \mathsf{F.Ev}(fk_s, g^x)$
$s \leftarrow \mathsf{H.Ev}(hk, i_s \,\|\, \sigma \,\|\, g^x \,\|\, g^{xy})$
If $i_s + 1 = j$ then $s \leftarrow rand$
$upd \leftarrow (g^x, \sigma)$; $\mathsf{auth}[i_s] \leftarrow upd$
$i_s \leftarrow i_s + 1$; $k_s \leftarrow s[1 \ldots \mathsf{RKE.kl}]$
$fk_s \leftarrow s[\mathsf{RKE.kl}+1 \ldots \mathsf{RKE.kl}+\mathsf{F.kl}]$
Return upd

$\textsc{ExpSim}$

If $\mathsf{op}[i_s] =$ “ch” then return $\bot$
$\mathsf{op}[i_s] \leftarrow$ “exp”
Return $(x, (i_s, fk_s), k_s)$

$\textsc{ChSendSim}$

If $\mathsf{op}[i_s] =$ “exp” then return $\bot$
$\mathsf{op}[i_s] \leftarrow$ “ch”
If $\mathsf{rkey}[i_s] = \bot$ then
  $\mathsf{rkey}[i_s] \leftarrow\$ \{0,1\}^{\mathsf{RKE.kl}}$
If $b = 1$ then return k_s
Else return $\mathsf{rkey}[i_s]$

$\textsc{ChRecSim}$

If $\mathsf{restricted}$ then return k_r
If $\mathsf{op}[i_r] =$ “exp” then return $\bot$
$\mathsf{op}[i_r] \leftarrow$ “ch”
If $\mathsf{rkey}[i_r] = \bot$ then
  $\mathsf{rkey}[i_r] \leftarrow\$ \{0,1\}^{\mathsf{RKE.kl}}$
If $b = 1$ then return k_r
Else return $\mathsf{rkey}[i_r]$

Figure 11: Adversary $\mathcal{F}$ for proof of Theorem 4.1.

$$= \Pr[\mathrm{G}_{1,q_{\textsc{RatSend}}}] + q_{\textsc{RatSend}} \cdot \mathsf{Adv}^{\mathsf{odhe}}_{\mathbb{G},\mathsf{H},\mathcal{O}_1} + \textstyle\sum_j \Pr[\mathsf{bad}^{\mathrm{I}_j}]$$
$$\leq q_{\textsc{RatSend}} \cdot \mathsf{Adv}^{\mathsf{odhe}}_{\mathbb{G},\mathsf{H},\mathcal{O}_1} + (q_{\textsc{RatSend}} + 1) \cdot \mathsf{Adv}^{\mathsf{sufcma}}_{\mathsf{F},\mathcal{F}} + \Pr[\mathrm{G}_{1,q_{\textsc{RatSend}}}].$$

Game G_1 is identical to $\mathrm{G}_{0,q_{\textsc{RatSend}}+1}$, but has been rewritten to allow make the final game transition of our proof easier to follow. The complicated, nested if-condition at the beginning of $\textsc{RatRec}$ has been simplified because $i_r < q_{\textsc{RatSend}} + 1$ always holds when $\mathsf{unchanged}$ is true. Additionally, when $\mathsf{unchanged}$ is true (and thus upd has been directly forwarded between $\textsc{RatSend}$ and $\textsc{RatRec}$ without being modified) we delay setting k_r, fk_r until they are about to be used, at which point they are set to match the appropriate k_s, fk_s that have been stored in a table. We have $\Pr[\mathrm{G}_{0,q_{\textsc{RatSend}}+1}] = \Pr[\mathrm{G}_1]$.

Games G_1 and G_2 differ only in that, in G_2, values of k_0 and fk_s are chosen at random instead of as the output of H (unless $\textsc{Exp}$ is called in which case we reset them to the correct output of H). We bound the difference between $\Pr[\mathrm{G}_1]$ and $\Pr[\mathrm{G}_2]$ by the advantage of the Diffie-Hellman adversary $\mathcal{O}_2$ that is defined in Fig. 12. Let b_{odhe} denote the challenge bit in game $\mathrm{ODHE}^{\mathcal{O}_2}_{\mathbb{G},\mathsf{H}}$, and

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

$b \leftarrow_\$ \{0,1\}$; $i_s \leftarrow 0$; $i_r \leftarrow 0$; $\mathsf{unchanged} \leftarrow \mathsf{true}$
$k_s[0] \leftarrow_\$ \{0,1\}^{\mathsf{RKE.kl}}$; $k_r \leftarrow k_s[0]$
$fk_s[0] \leftarrow_\$ \{0,1\}^{\mathsf{F.kl}}$; $fk_r \leftarrow fk_s[0]$; $hk \leftarrow_\$ \{0,1\}^{\mathsf{H.kl}}$
$g \leftarrow_\$ \mathbb{G}^*$; $y \leftarrow_\$ \mathbb{Z}_p$; $stk_s \leftarrow (hk, g, Y)$
$b' \leftarrow_\$ \mathcal{D}^{\text{RatSendSim},\text{RatRecSim},\text{ExpSim},\text{ChSendSim},\text{ChRecSim}}(stk_s)$
If $(b' = b)$ then return 1 else return 0

RatSendSim

If $\mathsf{op}[i_s] = \perp$ then
  $\mathsf{op}[i_s] \leftarrow$ "ch"
  If $i_s \neq 0$ then
    $(X, \sigma) \leftarrow \mathsf{auth}[i_s - 1]$; $s \leftarrow \text{Ch}(\sigma || X)$
    SaveKeys(i_s, s)
$X \leftarrow \text{Up}$; $\sigma \leftarrow \mathsf{F.Ev}(fk_s[i_s], X)$; $upd \leftarrow (X, \sigma)$
$\mathsf{auth}[i_s] \leftarrow upd$; $i_s \leftarrow i_s + 1$; Return upd

RatRecSim(upd)

$(X, \sigma) \leftarrow upd$
$\mathsf{forge} \leftarrow ((\mathsf{op}[i_r] \neq \text{"exp"}) \wedge (upd \neq \mathsf{auth}[i_r]))$
If $\mathsf{unchanged}$ and forge then return false
If $\mathsf{unchanged}$ then $fk_r \leftarrow fk_s[i_r]$
If $(\sigma \neq \mathsf{F.Ev}(fk_r, X))$ then return false
If $\mathsf{op}[i_r] =$ "exp" then $\mathsf{restricted} \leftarrow \mathsf{true}$
If $upd = \mathsf{auth}[i_r]$
  $\mathsf{unchanged} \leftarrow \mathsf{true}$; $\mathsf{restricted} \leftarrow \mathsf{false}$; $i_r \leftarrow i_r + 1$
Else
  $\mathsf{unchanged} \leftarrow \mathsf{false}$; $s \leftarrow \text{Hash}(i_r, \sigma || X, X)$
  $i_r \leftarrow i_r + 1$; $k_r \leftarrow s[1 \ldots \mathsf{RKE.kl}]$
  $fk_r \leftarrow s[\mathsf{RKE.kl} + 1 \ldots \mathsf{RKE.kl} + \mathsf{F.kl}]$
Return true

SaveKeys(i, s)

$k_s[i] \leftarrow s[1 \ldots \mathsf{RKE.kl}]$
$fk_s[i] \leftarrow s[\mathsf{RKE.kl} + 1 \ldots \mathsf{RKE.kl} + \mathsf{F.kl}]$

ExpSim

If $\mathsf{op}[i_s] =$ "ch" then return $\perp$
If $(\mathsf{op}[i_s] = \perp)$ and $(i_s \neq 0)$ then
  $x \leftarrow \text{Exp}$
  $(X, \sigma) \leftarrow \mathsf{auth}[i_s - 1]$
  $s \leftarrow \text{Hash}(i_s - 1, \sigma || X, X)$
  SaveKeys(i_s, s)
$\mathsf{op}[i_s] \leftarrow$ "exp"
Return $(x, (i_s, fk_s[i_s]), k_s[i_s])$

ChSendSim

If $\mathsf{op}[i_s] =$ "exp" then return $\perp$
If $(\mathsf{op}[i_s] = \perp)$ and $(i_s \neq 0)$ then
  $(X, \sigma) \leftarrow \mathsf{auth}[i_s - 1]$
  $s \leftarrow \text{Ch}(\sigma || X)$
  SaveKeys(i_s, s)
$\mathsf{op}[i_s] \leftarrow$ "ch"
If $\mathsf{rkey}[i_s] = \perp$ then
  $\mathsf{rkey}[i_s] \leftarrow_\$ \{0,1\}^{\mathsf{RKE.kl}}$
If $b = 1$ then return $k_s[i_s]$
Else return $\mathsf{rkey}[i_s]$

ChRecSim

If $\mathsf{restricted}$ then return k_r
If $\mathsf{op}[i_r] =$ "exp" then return $\perp$
If $(\mathsf{op}[i_r] = \perp)$ and $(i_r \neq 0)$ then
  $(X, \sigma) \leftarrow \mathsf{auth}[i_r - 1]$
  $s \leftarrow \text{Ch}(\sigma || X)$
  SaveKeys(i_r, s)
$\mathsf{op}[i_r] \leftarrow$ "ch"
If $\mathsf{rkey}[i_r] = \perp$ then
  $\mathsf{rkey}[i_r] \leftarrow_\$ \{0,1\}^{\mathsf{RKE.kl}}$
If $\mathsf{unchanged}$ then $k_r \leftarrow k_s[i_r]$
If $b = 1$ then return k_r
Else return $\mathsf{rkey}[i_r]$

Figure 12: Adversary $\mathcal{O}_2$ for proof of Theorem 4.1.

let b' denote the corresponding guess made by the adversary $\mathcal{O}_2$. $\mathcal{O}_2$ uses its own oracle to simulate the view of $\mathcal{D}$. When $b_{\mathsf{odhe}} = 1$ it perfectly simulates the view of $\mathcal{D}$ in G_1, and when $b_{\mathsf{odhe}} = 0$ it perfectly simulates the view of G_2. When $\mathcal{D}$ correctly guess the bit b then $\mathcal{O}_2$ assume its challenge oracle must returning real output from the hash function so it outputs $b' = 1$. Otherwise it outputs $b' = 0$. Thus, $\Pr[\text{G}_1] = \Pr[\, b' = 1 \,|\, b_{\mathsf{odhe}} = 1 \,]$ and $\Pr[\text{G}_2] = \Pr[\, b' = 1 \,|\, b_{\mathsf{odhe}} = 0 \,]$ from which it follows that $\mathsf{Adv}^{\mathsf{odhe}}_{\mathbb{G},\mathsf{H},\mathcal{O}_2} = \Pr[\text{G}_1] - \Pr[\text{G}_2]$. As a result of the above and our previous sequence of inequalities, we get:

$$\begin{aligned}\Pr[\text{KIND}^{\mathcal{D}}_{\mathsf{RKE}}] &\leq q_{\text{RatSend}} \cdot \mathsf{Adv}^{\mathsf{odhe}}_{\mathbb{G},\mathsf{H},\mathcal{O}_1} + (q_{\text{RatSend}} + 1) \cdot \mathsf{Adv}^{\mathsf{sufcma}}_{\mathsf{F},\mathcal{F}} + \Pr[\text{G}_1] \\ &= q_{\text{RatSend}} \cdot \mathsf{Adv}^{\mathsf{odhe}}_{\mathbb{G},\mathsf{H},\mathcal{O}_1} + (q_{\text{RatSend}} + 1) \cdot \mathsf{Adv}^{\mathsf{sufcma}}_{\mathsf{F},\mathcal{F}} + \mathsf{Adv}^{\mathsf{odhe}}_{\mathbb{G},\mathsf{H},\mathcal{O}_2} + \Pr[\text{G}_2].\end{aligned}$$

Finally, $\Pr[\text{G}_2] = 1/2$ because the view of $\mathcal{D}$ is independent of b in G_2. To see this, first note that oracle ChSend returns uniformly random bits regardless of the challenge bit. So we only need to

verify that the ChRec returns the same random bits if its last if-statement is reached. This could only fail to occur if ChRec was called when restricted and unchanged are both false. However, flags restricted and unchanged can only be simultaneously false at the end of an oracle call to RatRec if they were already both false at the time when this oracle was called. Thus no call to RatRec can be the first to set them both to false.

This yields the claimed bound on the advantage of $\mathcal{D}$. The bounds on the number of oracle queries made by the adversaries are obtained by examining their code.▮

5 Ratcheted encryption

In this section we define ratcheted encryption schemes, and show how to construct them by composing ratcheted key exchange with symmetric encryption. This serves as a starting point for discussing ratcheted encryption, and we also discuss possible extensions.

Ratcheted encryption schemes. Our definition of ratcheted encryption extends the definition of ratcheted key exchange by adding encryption and decryption algorithms. Ratcheted encryption schemes inherit the key generation algorithms from ratcheted key exchange schemes, and use the resulting shared keys as symmetric encryption keys. In line with our definition for ratcheted key exchange, we only consider one-sided ratcheted encryption, meaning that the sender uses its key only for encryption, and the receiver uses its key only for decryption.

A ratcheted encryption scheme RE specifies algorithms RE.IKg, RE.SKg, RE.RKg, RE.Enc and RE.Dec, where RE.Enc and RE.Dec are deterministic. Associated to RE is a nonce space RE.NS, sender's key generation randomness space RE.RS, and a ciphertext length function $\mathsf{RE.cl}\colon \mathbb{N} \to \mathbb{N}$. Initial key generation algorithm RE.IKg returns $k, sek_s, (stk_s, stk_r, sek_r)$, where k is an encryption key, stk_s, sek_s are a sender's static key and session key, and stk_r, sek_r are receiver's static key and receiver's session key, respectively. The sender's and receiver's (symmetric) encryption keys are initialized to $k_s = k_r = k$. Sender's key generation algorithm RE.SKg takes stk_s, sek_s and randomness $r \in \mathsf{RE.RS}$ to return a new sender's session key sek_s, a new sender's encryption key k_s, and update information upd. Receiver's key generation algorithm RE.RKg takes stk_s, stk_r, sek_r, upd and receiver's encryption key k_r to return a new receiver's session key sek_r, a new receiver's encryption key k_r, and a flag $acc \in \{\mathsf{true}, \mathsf{false}\}$. Encryption algorithm RE.Enc takes k_s, a nonce $n \in \mathsf{RE.NS}$, a plaintext message $m \in \{0,1\}^*$ and a header $h \in \{0,1\}^*$ to return a ciphertext $c \in \{0,1\}^{\mathsf{RE.cl}(|m|)}$. Decryption algorithm RE.Dec takes k_r, n, c, h to return $m \in \{0,1\}^* \cup \{\perp\}$.

Correctness of ratcheted encryption. Correctness of ratcheted encryption extends that of ratcheted key exchange. It requires that messages encrypted using sender's key should correctly decrypt using the corresponding receiver's key.

Consider game RE-COR of Fig. 4 associated to a ratcheted encryption scheme R and an adversary $\mathcal{C}$, where $\mathcal{C}$ is provided with an access to oracles Up, RatRec and Enc. The advantage of $\mathcal{C}$ breaking the correctness of R is defined as $\mathsf{Adv}^{\mathsf{recor}}_{\mathsf{R},\mathcal{C}} = 1 - \Pr[\text{RE-COR}^{\mathcal{C}}_{\mathsf{R}}]$. Correctness property requires that $\mathsf{Adv}^{\mathsf{recor}}_{\mathsf{R},\mathcal{C}} = 0$ for all unbounded adversaries $\mathcal{C}$. Compared to the correctness game for ratcheted key exchange, the new element is that adversary $\mathcal{C}$ also gets access to an encryption oracle Enc, which can be queried to test the decryption correctness.

Ratcheted authenticated encryption. Consider game RAE on the right side of Fig. 5 associated to a ratcheted encryption scheme RE and an adversary $\mathcal{A}$. It extends the security definition of ratcheted

Algorithm $\mathsf{RE.IKg}$

$(k, sek_s, (stk_s, stk_r, sek_r)) \leftarrow\$ \mathsf{RKE.IKg}$
Return $(k, sek_s, (stk_s, stk_r, sek_r))$

Algorithm $\mathsf{RE.SKg}(stk_s, sek_s; r)$

$(sek_s, k_s, upd) \leftarrow \mathsf{RKE.SKg}(stk_s, sek_s; r)$
Return (sek_s, k_s, upd)

Algorithm $\mathsf{RE.RKg}(stk_s, stk_r, sek_r, upd, k_r)$

$(sek_r, k_r, acc) \leftarrow\$ \mathsf{RKE.RKg}(stk_s, stk_r, sek_r, upd, k_r)$
Return (sek_r, k_r, acc)

Algorithm $\mathsf{RE.Enc}(k_s, n, m, h)$

$c \leftarrow \mathsf{SE.Enc}(k_s, n, m, h)$
Return c

Algorithm $\mathsf{RE.Dec}(k_r, n, c, h)$

$m \leftarrow \mathsf{SE.Dec}(k_r, n, c, h)$
Return m

Figure 13: Ratcheted encryption scheme $\mathsf{RE} = \mathsf{RATCHET\text{-}ENC}[\mathsf{RKE}, \mathsf{SE}]$.

key exchange (as defined in game KIND on the left side of Fig. 5) by replacing oracles ChSend and ChRec with oracles Enc and Dec. Oracles RatSend, RatRec and Exp are the same in both games. Oracles Enc and Dec are defined as follows. In the real world (when $b = 1$) oracle Enc encrypts messages under the sender's key, and oracle Dec decrypts ciphertexts under the receiver's key. In the random world (when $b = 0$) oracle Enc returns uniformly random strings, and oracle Dec always returns an incorrect decryption symbol $\perp$. The goal of the adversary is to distinguish between the two cases. The advantage of $\mathcal{A}$ in breaking the RAE security of RE is defined as $\mathsf{Adv}^{\mathsf{rae}}_{\mathsf{RE},\mathcal{A}} = 2\Pr[\mathrm{RAE}^{\mathcal{A}}_{\mathsf{RE}}] - 1$.

We note that the adversary is only allowed to get a single encryption for each unique pair of (i_s, n). This restriction stems from the fact that most known nonce-based encryption schemes are not resistant to *nonce-misuse*. Our definition can be relaxed to only prevent queries where (i_s, n, m) — or even (i_s, n, m, h) — are repeated, but it would increasingly limit the choice of the underlying symmetric schemes that can be used for this purpose (fewer schemes would satisfy stronger security definitions of multi-user authenticated encryption).

Revisiting the treatment of the restricted flag. Similar to the definition of KIND, one could consider strengthening the definition of RAE by never resetting the restricted flag back to false (as discussed in Section 4.2). There would seem to be a more clear motivation to use the stronger definition in the case of encryption. Namely, our current security definition allows adversary to comprimse the sender, use the exposed secrets to communicate with the receiver, and then restore the inital conversation link between the sender and the receiver. This represents an ability to stealthily insert arbitrary messages in the middle of someone's conversation, without ultimately disrupting the conversation. However, note that even a stonger definition (one that does not reset the restricted flag) appears to allow such attack, because the adversary might be able to compromise the sender and insert the messages before the next time the key ratcheting happens. The success of such attack would depend on how often the keys are being ratcheted.

Ratcheted encryption scheme RATCHET-ENC. We build a ratcheted encryption scheme by combining a ratcheted key exchange scheme with a symmetric encryption scheme. In our composition the output keys of the ratcheted key exchange scheme are used as encryption keys for the symmetric encryption scheme.

Let RKE be a ratcheted key exchange scheme. Let SE be a symmetric encryption scheme such that $\mathsf{SE.kl} = \mathsf{RKE.kl}$. We build a ratcheted encryption scheme $\mathsf{RE} = \mathsf{RATCHET\text{-}ENC}[\mathsf{RKE}, \mathsf{SE}]$ as defined in Fig. 13, with $\mathsf{RE.NS} = \mathsf{SE.NS}$, $\mathsf{RE.RS} = \mathsf{RKE.RS}$ and $\mathsf{RE.cl} = \mathsf{SE.cl}$.

Security of ratcheted encryption scheme RATCHET-ENC. The following says that the RAE security of ratcheted encryption scheme $\mathsf{RE} = \mathsf{RATCHET\text{-}ENC}[\mathsf{RKE}, \mathsf{SE}]$ can be reduced to the KIND security of the ratcheted key exchange scheme RKE and MAE security of the symmetric encryption scheme SE.

Theorem 5.1 *Let* RKE *be a ratcheted key exchange scheme. Let* SE *be a symmetric encryption scheme such that* $\mathsf{SE.kl} = \mathsf{RKE.kl}$. *Let* $\mathsf{RE} = \mathsf{RATCHET\text{-}ENC}[\mathsf{RKE}, \mathsf{SE}]$. *Let* $\mathcal{A}$ *be an adversary attacking the* RAE*-security of* RE *that makes* q_{RatSend} *queries to its* RatSend *oracle,* q_{RatRec} *queries to its* RatRec *oracle,* q_{Exp} *queries to its* Exp *oracle,* q_{Enc} *queries to its* Enc *oracle, and* q_{Dec} *queries to its* Dec *oracle. Then there is an adversary* $\mathcal{D}$ *attacking the* KIND*-security of* RKE *and an adversary* $\mathcal{N}$ *attacking the* MAE*-security of* SE *such that*

$$\mathsf{Adv}^{\mathsf{rae}}_{\mathsf{RE},\mathcal{A}} \leq 2 \cdot \mathsf{Adv}^{\mathsf{kind}}_{\mathsf{RKE},\mathcal{D}} + \mathsf{Adv}^{\mathsf{mae}}_{\mathsf{SE},\mathcal{N}}.$$

Adversary $\mathcal{D}$ *makes at most* q_{Exp} *queries to its* Exp *oracle,* q_{Enc} *queries to its* ChSend *oracle,* q_{Dec} *queries to its* ChRec *oracle, and the same number of queries as* $\mathcal{A}$ *to oracles* RatSend, RatRec. *Adversary* $\mathcal{N}$ *makes at most* $\max(q_{\text{RatSend}}, q_{\text{RatRec}})$ *queries to its* New *oracle,* q_{Enc} *queries to its* Enc *oracle, and* q_{Dec} *queries to its* Dec *oracle. Each of* $\mathcal{D}$, $\mathcal{N}$ *has a running time approximately that of* $\mathcal{A}$.

The proof is given below. It proceeds in two steps, first using the KIND-security of RKE and then using the MAE-security of SE. Recall that our goal is to show that adversary $\mathcal{A}$ playing RAE security game against RE is unable to distinguish between the real world (when oracles Enc and Dec return real encryptions and decryptions) and the random world (when oracle Enc returns random strings, and oracle Dec returns incorrect decryption symbol). In the first step of the proof, we use the KIND-security of ratcheted key exchange scheme RKE to switch from using real keys to random keys when calling oracles Enc and Dec in the real world of the RAE security game. We note that oracles RatSend, RatRec, Exp will still operate on the *real* keys after the first step, and the adversary $\mathcal{D}$ against KIND-security of RKE is able to simulate them using its own oracles. At this point, the keys used to answer queries to oracles Enc and Dec (in the game derived from the initial RAE security game) are random and independent of the keys used to answer queries to oracles RatSend, RatRec, Exp. Thus for the second step of the proof, we can build an adversary $\mathcal{N}$ against the MAE-security of SE that will generate its own set of keys for ratcheted encryption scheme RE and use them to produce simulated answers for $\mathcal{A}$'s oracle queries to RatSend, RatRec, Exp. Adversary $\mathcal{N}$ will answer $\mathcal{A}$'s oracle queries to Enc, Dec using the oracles provided by the MAE game, and relay $\mathcal{A}$'s output bit as the answer for its own security game.

Proof of Theorem 5.1: Consider games $\mathrm{G}_0, \mathrm{G}_1$ of Fig. 14. Lines not annotated with comments are common to both games. Game G_0 is equivalent to $\mathrm{RAE}^{\mathcal{A}}_{\mathsf{RE}}$, so

$$\mathsf{Adv}^{\mathsf{rae}}_{\mathsf{RE},\mathcal{A}} = 2\Pr[\mathrm{G}_0] - 1. \tag{2}$$

Game G_1 differs from game G_0 by using uniformly random keys to answer Enc and Dec oracle queries. Both games use real keys to answer Exp oracle queries.

First, we construct an adversary $\mathcal{D}$ against the KIND-security of RKE, as defined in Fig. 15. Adversary $\mathcal{D}$ simulates adversary $\mathcal{A}$ as follows. $\mathcal{A}$'s oracle queries to RatSend, RatRec and Exp are directly answered by the corresponding $\mathcal{D}$'s oracles (but $\mathcal{D}$ also does some bookkeeping to maintain the states that are necessary for simulating other oracle queries). $\mathcal{D}$ simulates $\mathcal{A}$'s

Games G_0–G_1

$(k, sek_s, (stk_s, stk_r, sek_r)) \leftarrow\$ \mathsf{RKE.IKg}$
$b \leftarrow\$ \{0,1\}$; $i_s \leftarrow 0$; $i_r \leftarrow 0$; $k_s \leftarrow k$; $k_r \leftarrow k$
$b' \leftarrow\$ \mathcal{A}^{\textsc{RatSend},\textsc{RatRec},\textsc{Exp},\textsc{Enc},\textsc{Dec}}(stk_s)$; Return $(b' = b)$

$\textsc{RatSend}$
$r \leftarrow\$ \mathsf{RKE.RS}$; $(sek_s, k_s, upd) \leftarrow \mathsf{RKE.SKg}(stk_s, sek_s; r)$
$\mathsf{auth}[i_s] \leftarrow upd$; $i_s \leftarrow i_s + 1$; Return upd

$\textsc{RatRec}(upd)$
$(sek_r, k_r, acc) \leftarrow\$ \mathsf{RKE.RKg}(stk_s, stk_r, sek_r, upd, k_r)$
If not acc then return false
If $\mathsf{op}[i_r] =$ "exp" then $\mathsf{restricted} \leftarrow \mathsf{true}$
If $upd = \mathsf{auth}[i_r]$ then $\mathsf{restricted} \leftarrow \mathsf{false}$
$i_r \leftarrow i_r + 1$; Return true

$\textsc{Exp}$
If $\mathsf{op}[i_s] =$ "ch" then return $\perp$
$\mathsf{op}[i_s] \leftarrow$ "exp" ; Return (r, sek_s, k_s)

$\textsc{Enc}(n, m, h)$
If $\mathsf{op}[i_s] =$ "exp" then return $\perp$
$\mathsf{op}[i_s] \leftarrow$ "ch"
If $(i_s, n) \in U$ then return $\perp$
$c_1 \leftarrow \mathsf{SE.Enc}(k_s, n, m, h)$ // G_0
If $\mathsf{rkey}[i_s] = \perp$ then $\mathsf{rkey}[i_s] \leftarrow\$ \{0,1\}^{\mathsf{RKE.kl}}$ // G_1
$c_1 \leftarrow \mathsf{SE.Enc}(\mathsf{rkey}[i_s], n, m, h)$ // G_1
$c_0 \leftarrow\$ \{0,1\}^{\mathsf{RE.cl}(|m|)}$; $U \leftarrow U \cup \{(i_s, n)\}$
$S \leftarrow S \cup \{(i_s, n, c_b, h)\}$; Return c_b

$\textsc{Dec}(n, c, h)$
If $\mathsf{restricted}$ then return $\mathsf{SE.Dec}(k_r, n, c, h)$
If $\mathsf{op}[i_r] =$ "exp" then return $\perp$
$\mathsf{op}[i_r] \leftarrow$ "ch"
If $(i_r, n, c, h) \in S$ then return $\perp$
$m \leftarrow \mathsf{SE.Dec}(k_r, n, c, h)$ // G_0
If $\mathsf{rkey}[i_r] = \perp$ then $\mathsf{rkey}[i_r] \leftarrow\$ \{0,1\}^{\mathsf{RKE.kl}}$ // G_1
$m \leftarrow \mathsf{SE.Dec}(\mathsf{rkey}[i_r], n, c, h)$ // G_1
If $b = 1$ then return m else return $\perp$

Figure 14: Games G_0, G_1 for proof of Theorem 5.1.

queries to Enc and Dec by calling it's own oracles ChSend and ChRec and using the received challenge keys to encrypt and decrypt the messages itself. Let b denote the challenge bit in game $\mathrm{KIND}^{\mathcal{D}}_{\mathsf{RKE}}$, and let b' denote the corresponding guess made by the adversary $\mathcal{D}$. We have $\Pr[G_0] = \Pr[\, b' = 1 \,|\, b = 1 \,]$ and $\Pr[G_1] = \Pr[\, b' = 1 \,|\, b = 0 \,]$. It follows that

$$\Pr[G_0] - \Pr[G_1] = \mathsf{Adv}^{\mathsf{kind}}_{\mathsf{RKE},\mathcal{D}}. \tag{3}$$

Next, we construct an adversary $\mathcal{N}$ against the MAE-security of SE, as defined in Fig. 16. Adversary $\mathcal{N}$ generates its own keys for the ratcheted key exchange scheme RKE, and uses them to answer $\mathcal{A}$'s queries to oracles RatSend, RatRec and Exp (as well as $\mathcal{A}$'s queries to Dec in the case when $\mathsf{restricted}$ is true). Furthermore, $\mathcal{A}$'s calls to Enc and Dec are answered using the corresponding

Adversary $\mathcal{D}^{\textsc{RatSend},\textsc{RatRec},\textsc{Exp},\textsc{ChSend},\textsc{ChRec}}(stk_s)$

$b \leftarrow_\$ \{0,1\}$; $i_s \leftarrow 0$; $i_r \leftarrow 0$
$b' \leftarrow_\$ \mathcal{A}^{\textsc{RatSendSim},\textsc{RatRecSim},\textsc{ExpSim},\textsc{EncSim},\textsc{DecSim}}(stk_s)$
If $(b' = b)$ then return 1 else return 0

$\textsc{EncSim}(n, m, h)$

If $\mathsf{op}[i_s] =$ "exp" then return $\perp$
$\mathsf{op}[i_s] \leftarrow$ "ch"
If $(i_s, n) \in U$ then return $\perp$
$k_s \leftarrow \textsc{ChSend}$; $c_1 \leftarrow \mathsf{SE.Enc}(k_s, n, m, h)$
$c_0 \leftarrow_\$ \{0,1\}^{\mathsf{RE.cl}(|m|)}$; $U \leftarrow U \cup \{(i_s, n)\}$
$S \leftarrow S \cup \{(i_s, n, c_b, h)\}$; Return c_b

$\textsc{DecSim}(n, c, h)$

If $\mathsf{restricted}$ then
  $k_r \leftarrow \textsc{ChRec}$; Return $\mathsf{SE.Dec}(k_r, n, c, h)$
If $\mathsf{op}[i_r] =$ "exp" then return $\perp$
$\mathsf{op}[i_r] \leftarrow$ "ch"
If $(i_r, n, c, h) \in S$ then return $\perp$
$k_r \leftarrow \textsc{ChRec}$; $m \leftarrow \mathsf{SE.Dec}(k_r, n, c, h)$
If $b = 1$ then return m else return $\perp$

$\textsc{RatSendSim}$

$upd \leftarrow \textsc{RatSend}$
$\mathsf{auth}[i_s] \leftarrow upd$
$i_s \leftarrow i_s + 1$
Return upd

$\textsc{RatRecSim}(upd)$

$\mathsf{success} \leftarrow \textsc{RatRec}(upd)$
If $\mathsf{success}$ then
  $u_0 \leftarrow (\mathsf{op}[i_r] =$ "exp"$)$
  $u_1 \leftarrow (upd = \mathsf{auth}[i_r])$
  If u_0 then $\mathsf{restricted} \leftarrow \mathsf{true}$
  If u_1 then $\mathsf{restricted} \leftarrow \mathsf{false}$
  $i_r \leftarrow i_r + 1$
Return $\mathsf{success}$

$\textsc{ExpSim}$

If $\mathsf{op}[i_s] =$ "ch" then return $\perp$
$\mathsf{op}[i_s] \leftarrow$ "exp"
Return $\textsc{Exp}$

Figure 15: Adversary $\mathcal{D}$ for proof of Theorem 5.1.

Adversary $\mathcal{N}^{\textsc{New},\textsc{Enc},\textsc{Dec}}$

$(k, sek_s, (stk_s, stk_r, sek_r)) \leftarrow_\$ \mathsf{RKE.IKg}$
$v \leftarrow 0$; $i_s \leftarrow 0$; $i_r \leftarrow 0$; $k_s \leftarrow k$; $k_r \leftarrow k$
$b' \leftarrow_\$ \mathcal{A}^{\textsc{RatSendSim},\textsc{RatRecSim},\textsc{ExpSim},\textsc{EncSim},\textsc{DecSim}}(stk_s)$
Return b'

$\textsc{RatSendSim}$

$r \leftarrow_\$ \mathsf{RKE.RS}$; $z \leftarrow \mathsf{RKE.SKg}(stk_s, sek_s; r)$
$(sek_s, k_s, upd) \leftarrow z$; $\mathsf{auth}[i_s] \leftarrow upd$; $i_s \leftarrow i_s + 1$
While $v < i_s$ do
  $\textsc{New}$; $v \leftarrow v + 1$
Return upd

$\textsc{RatRecSim}(upd)$

$(sek_r, k_r, acc) \leftarrow_\$ \mathsf{RKE.RKg}(stk_s, stk_r, sek_r, upd, k_r)$
If not acc then return false
If $\mathsf{op}[i_r] =$ "exp" then $\mathsf{restricted} \leftarrow \mathsf{true}$
If $upd = \mathsf{auth}[i_r]$ then $\mathsf{restricted} \leftarrow \mathsf{false}$
$i_r \leftarrow i_r + 1$
While $v < i_r$ do
  $\textsc{New}$; $v \leftarrow v + 1$
Return true

$\textsc{ExpSim}$

If $\mathsf{op}[i_s] =$ "ch" then return $\perp$
$\mathsf{op}[i_s] \leftarrow$ "exp"
Return (r, sek_s, k_s)

$\textsc{Enc}(n, m, h)$

If $\mathsf{op}[i_s] =$ "exp" then return $\perp$
$\mathsf{op}[i_s] \leftarrow$ "ch"
Return $\textsc{Enc}(i_s, n, m, h)$

$\textsc{Dec}(n, c, h)$

If $\mathsf{restricted}$ then
  Return $\mathsf{SE.Dec}(k_r, n, c, h)$
If $\mathsf{op}[i_r] =$ "exp" then return $\perp$
$\mathsf{op}[i_r] \leftarrow$ "ch"
Return $\textsc{Dec}(i_r, n, c, h)$

Figure 16: Adversary $\mathcal{N}$ for proof of Theorem 5.1.

oracles that are provided to $\mathcal{N}$ in game MAE. We have $\Pr[\mathrm{G}_1] = \mathrm{MAE}^{\mathcal{N}}_{\mathsf{SE}}$, so

$$\mathsf{Adv}^{\mathsf{mae}}_{\mathsf{SE},\mathcal{N}} = 2\Pr[\mathrm{G}_1] - 1. \tag{4}$$

The theorem statement follows from equations (2)–(4). ∎

Extensions. We defined our encryption schemes to be one-sided in both communication (meaning that the messages are assumed to be sent only in one direction, from the sender to the receiver), and in security (only protecting against the exposure of the sender's secrets). It would be useful to consider *two-sided* communication (but still one-sided security). In our model the sender and the receiver already share the same key, but one would need to update the security game to allow using either key for encryption and decryption.

An important goal in studying ratcheted encryption is to model the *Double Ratchet algorithm* [19, 15] used in multiple real-world messaging applications, such as in WhatsApp [25] and in the Secret Conversations mode of Facebook Messenger [16]. This work models the asymmetric layer of key ratcheting, whereas the real-world applications also have a second layer of key ratcheting that happens in a symmetric setting. In our model, this can be possibly achieved by using the output keys of ratcheted key exchange to initialize a *forward-secure* symmetric encryption scheme. We do not capture this possibility; both the syntax and the security definitions would need to be significantly extended.

Acknowledgments

We thank the EUROCRYPT 2017 and CRYPTO 2017 reviewers for their comments.

References

[1] M. Abdalla, M. Bellare, and P. Rogaway. The oracle Diffie-Hellman assumptions and an analysis of DHIES. In D. Naccache, editor, *CT-RSA 2001*, volume 2020 of *LNCS*, pages 143–158, San Francisco, CA, USA, Apr. 8–12, 2001. Springer, Heidelberg, Germany. Full version at `http://web.cs.ucdavis.edu/~rogaway/papers/dhies.pdf`. 3, 4, 5, 7, 8, 9, 32

[2] M. Bellare, S. Duan, and A. Palacio. Key insulation and intrusion resilience over a public channel. In M. Fischlin, editor, *CT-RSA 2009*, volume 5473 of *LNCS*, pages 84–99, San Francisco, CA, USA, Apr. 20–24, 2009. Springer, Heidelberg, Germany. 5

[3] M. Bellare, D. Pointcheval, and P. Rogaway. Authenticated key exchange secure against dictionary attacks. In B. Preneel, editor, *EUROCRYPT 2000*, volume 1807 of *LNCS*, pages 139–155, Bruges, Belgium, May 14–18, 2000. Springer, Heidelberg, Germany. 5

[4] M. Bellare and P. Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In V. Ashby, editor, *ACM CCS 93*, pages 62–73, Fairfax, Virginia, USA, Nov. 3–5, 1993. ACM Press. 3, 32

[5] M. Bellare and P. Rogaway. Entity authentication and key distribution. In D. R. Stinson, editor, *CRYPTO'93*, volume 773 of *LNCS*, pages 232–249, Santa Barbara, CA, USA, Aug. 22–26, 1994. Springer, Heidelberg, Germany. 3, 4, 5

[6] M. Bellare and P. Rogaway. The security of triple encryption and a framework for code-based game-playing proofs. In S. Vaudenay, editor, *EUROCRYPT 2006*, volume 4004 of *LNCS*, pages 409–426, St. Petersburg, Russia, May 28 – June 1, 2006. Springer, Heidelberg, Germany. 6, 19, 36, 42

[7] M. Bellare and B. Tackmann. The multi-user security of authenticated encryption: AES-GCM in TLS 1.3. In M. Robshaw and J. Katz, editors, *CRYPTO 2016, Part I*, volume 9814 of *LNCS*, pages 247–276, Santa Barbara, CA, USA, Aug. 14–18, 2016. Springer, Heidelberg, Germany. 3, 4, 7

[8] M. Bellare and B. S. Yee. Forward-security in private-key cryptography. In M. Joye, editor, *CT-RSA 2003*, volume 2612 of *LNCS*, pages 1–18, San Francisco, CA, USA, Apr. 13–17, 2003. Springer, Heidelberg, Germany. 4

[9] N. Borisov, I. Goldberg, and E. Brewer. Off-the-record communication, or, why not to use pgp. In *Proceedings of the 2004 ACM Workshop on Privacy in the Electronic Society*, WPES '04, pages 77–84, New York, NY, USA, 2004. ACM. 2, 3, 4

[10] K. Cohn-Gordon, C. Cremers, B. Dowling, L. Garratt, and D. Stebila. A formal security analysis of the Signal messaging protocol. In *Proc. IEEE European Symposium on Security and Privacy (EuroS&P) 2017*. IEEE, April 2017. To appear. 5

[11] K. Cohn-Gordon, C. Cremers, and L. Garratt. On post-compromise security. In *2016 IEEE 29th Computer Security Foundations Symposium (CSF)*, pages 164–178, June 2016. 5

[12] Y. Dodis, J. Katz, S. Xu, and M. Yung. Key-insulated public key cryptosystems. In L. R. Knudsen, editor, *EUROCRYPT 2002*, volume 2332 of *LNCS*, pages 65–82, Amsterdam, The Netherlands, Apr. 28 – May 2, 2002. Springer, Heidelberg, Germany. 5

[13] Y. Dodis, J. Katz, S. Xu, and M. Yung. Strong key-insulated signature schemes. In Y. Desmedt, editor, *PKC 2003*, volume 2567 of *LNCS*, pages 130–144, Miami, FL, USA, Jan. 6–8, 2003. Springer, Heidelberg, Germany. 5

[14] Y. Dodis, W. Luo, S. Xu, and M. Yung. Key-insulated symmetric key cryptography and mitigating attacks against cryptographic cloud software. In H. Y. Youm and Y. Won, editors, *ASIACCS 12*, pages 57–58, Seoul, Korea, May 2–4, 2012. ACM Press. 5

[15] T. P. (editor) and M. Marlinspike. The double ratchet algorithm. `https://whispersystems.org/docs/specifications/doubleratchet/`, Nov. 20, 2016. Accessed: 2017-06-03. 16, 30

[16] Facebook. Messenger secret conversations technical whitepaper. Technical whitepaper, `https://fbnewsroomus.files.wordpress.com/2016/07/secret_conversations_whitepaper-1.pdf`, July 8, 2016. Accessed: 2016-09-15. 30

[17] M. Fischlin and F. Günther. Multi-stage key exchange and the case of Google's QUIC protocol. In G.-J. Ahn, M. Yung, and N. Li, editors, *ACM CCS 14*, pages 1193–1204, Scottsdale, AZ, USA, Nov. 3–7, 2014. ACM Press. 5

[18] A. Langley. Pond. GitHub repository, README.md, `https://github.com/agl/pond/commit/7bb06244b9aa121d367a6d556867992d1481f0c8`, 2012. Accessed: 2017-06-03. 2

[19] M. Marlinspike. Advanced cryptographic ratcheting. `https://whispersystems.org/blog/advanced-ratcheting/`, Nov. 26, 2013. Accessed: 2016-09-15. 16, 30

[20] T. Okamoto and D. Pointcheval. The gap-problems: A new class of problems for the security of cryptographic schemes. In K. Kim, editor, *PKC 2001*, volume 1992 of *LNCS*, pages 104–118, Cheju Island, South Korea, Feb. 13–15, 2001. Springer, Heidelberg, Germany. 4, 5, 8

[21] Open Whisper Systems. Signal protocol library for java/android. GitHub repository, `https://github.com/WhisperSystems/libsignal-protocol-java`, 2017. Accessed: 2017-06-03. 2, 16

[22] P. Rogaway. Authenticated-encryption with associated-data. In V. Atluri, editor, *ACM CCS 02*, pages 98–107, Washington D.C., USA, Nov. 18–22, 2002. ACM Press. 3, 4, 7

[23] P. Rogaway. Nonce-based symmetric encryption. In B. K. Roy and W. Meier, editors, *FSE 2004*, volume 3017 of *LNCS*, pages 348–359, New Delhi, India, Feb. 5–7, 2004. Springer, Heidelberg, Germany. 7

[24] N. Unger, S. Dechand, J. Bonneau, S. Fahl, H. Perl, I. Goldberg, and M. Smith. SoK: Secure messaging. In *2015 IEEE Symposium on Security and Privacy*, pages 232–249, San Jose, CA, USA, May 17–21, 2015. IEEE Computer Society Press. 2

[25] WhatsApp. Whatsapp encryption overview. Technical white paper, `https://www.whatsapp.com/security/WhatsApp-Security-Whitepaper.pdf`, Apr. 4, 2016. Accessed: 2016-09-15. 30

A Oracle Diffie-Hellman with Exposures in ROM

In this section we justify the plausibility of the Oracle Diffie-Hellman with Exposures (ODHE) assumption that was introduced in Section 3. We used ODHE to prove the security of our ratcheted key exchange scheme in Section 4.4. We now show that the ODHE assumption reduces to the Strong Computational Diffie-Hellman (SCDH) assumption when the hash function in the former is modeled as the random oracle.

We do the reduction in two steps. We introduce the *Strong Computational Diffie-Hellman with Exposures* (SCDHE) assumption and use it as an intermediate assumption. In the first step, we show that ODHE reduces to SCDHE when the hash function in ODHE is modeled as the random oracle. In the second step, we show that SCDHE reduces to SCDH. We provide two different reductions for the second step. Our first reduction uses a standard index guessing proof and thus creates a factor q loss in the advantage. Our second reduction avoids this multiplicative advantage loss by defining a "rewinding" adversary that uses the self-reducibility of Diffie-Hellman problems.

We now define the necessary assumptions and state two alternative theorems as outlined above.

Random Oracle Model. In the first step of our reduction we will work in the random oracle model (ROM) [4], modeling a hash function as the random oracle. The random oracle RO models a truly random function and is defined as follows:

$$\begin{array}{l} \underline{\mathrm{RO}(z,\kappa)} \\ \text{If } T[z,\kappa] = \bot \text{ then } T[z,\kappa] \leftarrow_{\$} \{0,1\}^{\kappa} \\ \text{Return } T[z,\kappa] \end{array}$$

It takes a string $z \in \{0,1\}^*$ and an output length $\kappa \in \mathbb{N}$ as input, to return an element from $\{0,1\}^{\kappa}$. We prove our claims for a hash function that simply evaluates the random oracle on its inputs. We now extend the ODHE assumption from Section 3 to the random oracle model.

Oracle Diffie-Hellman with Exposures in ROM assumption. Let $\mathbb{G}$ be a cyclic group of order $p \in \mathbb{N}$, and let $\mathbb{G}^*$ denote the set of its generators. Let H be a function family such that $\mathsf{H.In} = \{0,1\}^*$. Consider game ODHER of Fig. 17 associated to $\mathbb{G}, \mathsf{H}$ and an adversary $\mathcal{O}$, where $\mathcal{O}$ is required to call oracle Up at least once prior to making any oracle queries to Ch and Exp. This game is similar to the game ODHE from Section 3, except that it provides adversary $\mathcal{O}$ and hash function H with an access to the random oracle RO. The advantage of $\mathcal{O}$ in breaking the ODHER security of $\mathbb{G}, \mathsf{H}$ is defined as $\mathsf{Adv}^{\mathsf{odher}}_{\mathbb{G},\mathsf{H},\mathcal{O}} = 2\Pr[\mathrm{ODHER}^{\mathcal{O}}_{\mathbb{G},\mathsf{H}}] - 1$.

Strong Computational Diffie-Hellman assumption. Let $\mathbb{G}$ be a cyclic group of order $p \in \mathbb{N}$, and let $\mathbb{G}^*$ denote the set of its generators. Consider game SCDH of Fig. 18, associated to group $\mathbb{G}$ and to an adversary $\mathcal{S}$. Adversary $\mathcal{S}$ receives g, g^x, g^y as input, where g is a group generator and g^x, g^y are random group elements for some secret values x, y. It is also provided with an oracle DH that takes arbitrary group elements X, Z and returns whether $X^y = Z$. The adversary wins the game if it can compute the value of g^{xy}. The advantage of $\mathcal{S}$ in breaking the SCDH security of $\mathbb{G}$ is defined as $\mathsf{Adv}^{\mathsf{scdh}}_{\mathbb{G},\mathcal{S}} = \Pr[\mathrm{SCDH}^{\mathcal{S}}_{\mathbb{G}}]$. The SCDH assumption was originally defined in [1].

Strong Computational Diffie-Hellman with Exposures assumption. Let $\mathbb{G}$ be a cyclic group of order $p \in \mathbb{N}$, and let $\mathbb{G}^*$ denote the set of its generators. Consider game SCDHE of Fig. 18, associated to group $\mathbb{G}$ and to an adversary $\mathcal{B}$. Adversary $\mathcal{B}$ receives g, g^y as input, where g is a group generator and g^y is a random group element for some secret value y. It is also provided with oracles DH, Up, Exp defined as follows. The DH oracle is identical to the one from the SCDH game; it takes arbitrary

Game $\mathrm{ODHER}^{\mathcal{O}}_{\mathbb{G},\mathsf{H}}$

$b \leftarrow\$ \{0,1\}$; $hk \leftarrow\$ \{0,1\}^{\mathsf{H.kl}}$; $g \leftarrow\$ \mathbb{G}^*$; $y \leftarrow\$ \mathbb{Z}_p$; $v \leftarrow -1$
$b' \leftarrow\$ \mathcal{O}^{\textsc{Up},\textsc{Ch},\textsc{Exp},\textsc{Hash},\mathrm{RO}}(hk, g, g^y)$; Return $(b' = b)$

$\textsc{Up}$

$\mathsf{op} \leftarrow \varepsilon$; $v \leftarrow v + 1$; $x[v] \leftarrow\$ \mathbb{Z}_p$; Return $g^{x[v]}$

$\textsc{Ch}(s)$

If (op = "exp") or $((v, s, g^{x[v]}) \in S_{\mathsf{hash}})$ then return $\perp$
$\mathsf{op} \leftarrow$ "ch" ; $S_{\mathsf{ch}} \leftarrow S_{\mathsf{ch}} \cup \{(v, s, g^{x[v]})\}$; $e \leftarrow g^{x[v]\cdot y}$
If $\mathsf{mem}[v, s, e] = \perp$ then $\mathsf{mem}[v, s, e] \leftarrow\$ \{0,1\}^{\mathsf{H.ol}}$
$r_1 \leftarrow \mathsf{H.Ev}^{\mathrm{RO}}(hk, v \,\|\, s \,\|\, e)$; $r_0 \leftarrow \mathsf{mem}[v, s, e]$; Return r_b

$\textsc{Exp}$

If op = "ch" then return $\perp$
$\mathsf{op} \leftarrow$ "exp" ; Return $x[v]$

$\textsc{Hash}(i, s, X)$

If $(i, s, X) \in S_{\mathsf{ch}}$ then return $\perp$
If $i = v$ then $S_{\mathsf{hash}} \leftarrow S_{\mathsf{hash}} \cup \{(i, s, X)\}$
Return $\mathsf{H.Ev}^{\mathrm{RO}}(hk, i \,\|\, s \,\|\, X^y)$

$\mathrm{RO}(z, \kappa)$

If $T[z, \kappa] = \perp$ then $T[z, \kappa] \leftarrow\$ \{0,1\}^{\kappa}$
Return $T[z, \kappa]$

Figure 17: Game defining Oracle Diffie-Hellman with Exposures in ROM assumption for $\mathbb{G}, \mathsf{H}$.

Game $\mathrm{SCDH}^{\mathcal{S}}_{\mathbb{G}}$	Game $\mathrm{SCDHE}^{\mathcal{B}}_{\mathbb{G}}$
$g \leftarrow\$ \mathbb{G}^*$; $y \leftarrow\$ \mathbb{Z}_p$; $x \leftarrow\$ \mathbb{Z}_p$ $Z \leftarrow\$ \mathcal{S}^{\mathrm{DH}}(g, g^x, g^y)$ Return $(Z = g^{xy})$ $\mathrm{DH}(X, Z)$ Return $(X^y = Z)$	$g \leftarrow\$ \mathbb{G}^*$; $y \leftarrow\$ \mathbb{Z}_p$; $v \leftarrow -1$ $(j, Z) \leftarrow\$ \mathcal{B}^{\mathrm{DH},\textsc{Up},\textsc{Exp}}(g, g^y)$ $\mathsf{valid} \leftarrow (0 \le j \le v)$ and $(\mathsf{op}[j] \neq$ "exp") Return valid and $(Z = g^{x[j]\cdot y})$ $\mathrm{DH}(X, Z)$ Return $(X^y = Z)$ $\textsc{Up}$ $v \leftarrow v + 1$; $x[v] \leftarrow\$ \mathbb{Z}_p$; Return $g^{x[v]}$ $\textsc{Exp}$ $\mathsf{op}[v] \leftarrow$ "exp" ; Return $x[v]$

Figure 18: Games defining Strong Computational Diffie-Hellman assumption in group $\mathbb{G}$, and Strong Computational Diffie-Hellman with Exposures assumption in group $\mathbb{G}$.

---

group elements X, Z and returns whether $X^y = Z$. The update oracle $\textsc{Up}$ generates a new random group exponent $x[v]$ and returns $g^{x[v]}$, where v is a counter that enumerates all challenge exponents (indexed from 0). The expose oracle $\textsc{Exp}$ returns $x[v]$ (we do not require that $\mathcal{B}$ calls $\textsc{Up}$ prior to its first call to $\textsc{Exp}$, meaning that $\mathcal{B}$ is allowed to expose the unitialized value at the location -1 of map x). The adversary wins the game if it returns a pair (j, Z) such that $Z = g^{x[j]\cdot y}$ and $x[j]$ was not exposed. The advantage of $\mathcal{B}$ in breaking the SCDHE security of $\mathbb{G}$ is defined as $\mathsf{Adv}^{\mathsf{scdhe}}_{\mathbb{G},\mathcal{B}} = \Pr[\mathrm{SCDHE}^{\mathcal{B}}_{\mathbb{G}}]$.

Theorem statements. We prove the following theorems. They relate the advantage of an adversary $\mathcal{O}$ against the ODHER security of $\mathbb{G}, \mathsf{H}$ —where H is instantiated by the random oracle— to the advantage of an adversary $\mathcal{S}$ against the SCDH security in $\mathbb{G}$. The second theorem achieves a better upper-bound for the advantage of $\mathcal{O}$, but its proof is more involved and the constructed adversary $\mathcal{S}$ has a worse running time.

We prove our claims for a function family H defined by $\mathsf{H.Ev}^{\mathrm{RO}}(hk, s) = \mathrm{RO}(hk \,\|\, s, \mathsf{H.ol})$ for some $\mathsf{H.ol} \in \mathbb{N}$. Note that the evaluation algorithm of a function family is defined to be deterministic, whereas RO is a randomized procedure. For H to be well-defined, one has to be careful when defining deterministic algorithms. For our purposes, a deterministic algorithms is one that takes no random coins.

Theorem A.1 *Let $\mathbb{G}$ be a cyclic group of order $p \in \mathbb{N}$, and let $\mathbb{G}^*$ denote the set of its generators. Let H be a function family defined by $\mathsf{H.Ev}^{\mathrm{RO}}(hk, s) = \mathrm{RO}(hk \,\|\, s, \mathsf{H.ol})$ for some $\mathsf{H.ol} \in \mathbb{N}$. Let $\mathcal{O}$ be an adversary attacking the* ODHER*-security of $\mathbb{G}, \mathsf{H}$ that makes q_{UP} queries to its* UP *oracle, q_{CH} queries to its* CH *oracle, q_{HASH} queries to its* HASH *oracle, and q_{RO} queries to its* RO *oracle. Then there is an adversary $\mathcal{S}$ attacking the* SCDH*-security of $\mathbb{G}$ such that*

$$\mathsf{Adv}^{\mathsf{odher}}_{\mathbb{G},\mathsf{H},\mathcal{O}} \leq q_{\mathrm{UP}} \cdot \mathsf{Adv}^{\mathsf{scdh}}_{\mathbb{G},\mathcal{S}}.$$

Adversary $\mathcal{S}$ makes at most $q_{\mathrm{RO}} \cdot (q_{\mathrm{CH}} + q_{\mathrm{HASH}})$ queries to its DH *oracle. Its running time is approximately that of $\mathcal{O}$ plus extra terms that are dominated by $q_{\mathrm{RO}} \cdot (q_{\mathrm{CH}} + q_{\mathrm{HASH}})$.*

To prove this theorem, Appendix A.1 provides the following. First, it reduces ODHE to SCDHE where the hash function in the former is modeled as the random oracle. Second, it reduces SCDHE to SCDH in the standard model.

Theorem A.2 *Let $\mathbb{G}$ be a cyclic group of order $p \in \mathbb{N}$, and let $\mathbb{G}^*$ denote the set of its generators. Let H be a function family defined by $\mathsf{H.Ev}^{\mathrm{RO}}(hk, s) = \mathrm{RO}(hk \,\|\, s, \mathsf{H.ol})$ for some $\mathsf{H.ol} \in \mathbb{N}$. Let $\mathcal{O}$ be an adversary attacking the* ODHER*-security of $\mathbb{G}, \mathsf{H}$ that makes q_{UP} queries to its* UP *oracle, q_{CH} queries to its* CH *oracle, q_{EXP} queries to its* EXP *oracle, q_{HASH} queries to its* HASH *oracle, and q_{RO} queries to its* RO *oracle. Let $u \in \mathbb{N}$. Then there is an adversary $\mathcal{S}_u$ attacking the* SCDH*-security of $\mathbb{G}$ such that*

$$\mathsf{Adv}^{\mathsf{odher}}_{\mathbb{G},\mathsf{H},\mathcal{O}} \leq \mathsf{Adv}^{\mathsf{scdh}}_{\mathbb{G},\mathcal{S}_u} + q_{\mathrm{UP}} \cdot 2^{-u}.$$

Let $q_0 = q_{\mathrm{RO}} \cdot (q_{\mathrm{CH}} + q_{\mathrm{HASH}})$ and $q_1 = 1 + u \cdot q_{\mathrm{UP}}$. Adversary $\mathcal{S}_u$ makes at most $q_0 \cdot q_1$ queries to its DH *oracle. Its running time is approximately q_1 times that of $\mathcal{O}$ plus extra terms that are dominated by $q_0 \cdot q_1$.*

In Appendix A.2 we provide an alternative reduction from SCDHE to SCDH that is more involved than the one in Appendix A.1. The claims in Theorem A.2 are implied by the random oracle reduction from ODHE to SCDHE in Appendix A.1, along with the improved reduction from SCDHE to SCDH in Appendix A.2.

A.1 ODHE reduction to SCDH in ROM

We now state and prove two lemmas that together imply the claim in Theorem A.1. The first lemma reduces from ODHE to SCDHE in ROM, by showing how to use any ODHER adversary $\mathcal{O}$ to construct an SCDHE adversary $\mathcal{B}$. Then the second lemma reduces from any SCDHE adversary $\mathcal{B}$ to an adversary $\mathcal{S}$ against SCDH.

Games $\mathrm{G}_0, \mathrm{G}_1$

$hk \leftarrow\$ \{0,1\}^{\mathsf{H.kl}}$; $g \leftarrow\$ \mathbb{G}^*$; $y \leftarrow\$ \mathbb{Z}_p$; $v \leftarrow -1$
$b' \leftarrow\$ \mathcal{O}^{\textsc{Up},\textsc{Ch},\textsc{Exp},\textsc{Hash},\mathrm{RO}}(hk, g, g^y)$; Return $(b' = 1)$

$\textsc{Up}$

$\mathsf{op} \leftarrow \varepsilon$; $v \leftarrow v + 1$; $x[v] \leftarrow\$ \mathbb{Z}_p$; Return $g^{x[v]}$

$\textsc{Ch}(s)$

If (op = "exp") or $((v, s, g^{x[v]}) \in S_{\mathsf{hash}})$ then return $\perp$
$\mathsf{op} \leftarrow$ "ch" ; $S_{\mathsf{ch}} \leftarrow S_{\mathsf{ch}} \cup \{(v, s, g^{x[v]})\}$; $e \leftarrow g^{x[v]\cdot y}$
If $\mathsf{mem}[v, s, e] = \perp$ then $\mathsf{mem}[v, s, e] \leftarrow\$ \{0,1\}^{\mathsf{H.ol}}$
$z \leftarrow hk \,\|\, v \,\|\, s \,\|\, e$; $\kappa \leftarrow \mathsf{H.ol}$
If $(T[z, \kappa] \neq \perp)$ then
  $\mathsf{bad} \leftarrow \mathsf{true}$
  $\mathsf{mem}[v, s, e] \leftarrow T[z, \kappa]$ // G_1
Return $\mathsf{mem}[v, s, e]$

$\textsc{Exp}$

If op = "ch" then return $\perp$
$\mathsf{op} \leftarrow$ "exp" ; Return $x[v]$

$\textsc{Hash}(i, s, X)$

If $(i, s, X) \in S_{\mathsf{ch}}$ then return $\perp$
If $i = v$ then $S_{\mathsf{hash}} \leftarrow S_{\mathsf{hash}} \cup \{(i, s, X)\}$
Return $\mathrm{RO}(hk \,\|\, i \,\|\, s \,\|\, X^y, \mathsf{H.ol})$

$\mathrm{RO}(z, \kappa)$

If $T[z, \kappa] = \perp$ then $T[z, \kappa] \leftarrow\$ \{0,1\}^\kappa$
$hk' \,\|\, i \,\|\, s \,\|\, e \leftarrow z$
If $(hk' = hk)$ and $(\mathsf{mem}[i, s, e] \neq \perp)$ and $(\kappa = \mathsf{H.ol})$ then
  $\mathsf{bad} \leftarrow \mathsf{true}$
  $T[z, \kappa] \leftarrow \mathsf{mem}[i, s, e]$ // G_1
Return $T[z, \kappa]$

Figure 19: Games $\mathrm{G}_0, \mathrm{G}_1$ for proof of Lemma A.3.

Reducing ODHE to SCDHE in ROM. The first intermediate lemma is as follows.

Lemma A.3 *Let $\mathbb{G}$ be a cyclic group of order $p \in \mathbb{N}$, and let $\mathbb{G}^*$ denote the set of its generators. Let H be a function family defined by $\mathsf{H.Ev}^{\mathrm{RO}}(hk, s) = \mathrm{RO}(hk \,\|\, s, \mathsf{H.ol})$ for some $\mathsf{H.ol} \in \mathbb{N}$. Let $\mathcal{O}$ be an adversary attacking the* ODHER*-security of $\mathbb{G}, \mathsf{H}$ that makes $q_{\textsc{Up}}$ queries to its* Up *oracle, $q_{\textsc{Ch}}$ queries to its* Ch *oracle, $q_{\textsc{Exp}}$ queries to its* Exp *oracle, $q_{\textsc{Hash}}$ queries to its* Hash *oracle, and q_{RO} queries to its* RO *oracle. Then there is an adversary $\mathcal{B}$ attacking the* SCDHE*-security of $\mathbb{G}$ such that*

$$\mathsf{Adv}^{\mathsf{odher}}_{\mathbb{G},\mathsf{H},\mathcal{O}} \leq \mathsf{Adv}^{\mathsf{scdhe}}_{\mathbb{G},\mathcal{B}}.$$

Adversary $\mathcal{B}$ makes $q_{\textsc{Up}}$ queries to its Up *oracle, $q_{\textsc{Exp}}$ queries to its* Exp *oracle, and at most $q_{\mathrm{RO}} \cdot (q_{\textsc{Ch}} + q_{\textsc{Hash}})$ queries to its* DH *oracle. Its running time is approximately that of $\mathcal{O}$ plus extra terms that are dominated by $q_{\mathrm{RO}} \cdot (q_{\textsc{Ch}} + q_{\textsc{Hash}})$.*

Proof of Lemma A.3: Consider games $\mathrm{G}_0, \mathrm{G}_1$ of Fig. 19. Lines not annotated with comments

are common to both games. Let b_{odher} be the challenge bit in game $\mathrm{ODHER}^{\mathcal{O}}_{\mathbb{G},\mathsf{H}}$, and let b' be the corresponding guess made by the adversary $\mathcal{O}$. We claim that $\Pr[\mathrm{G}_1] = \Pr[\, b' = 1 \mid b_{\mathsf{odher}} = 1\,]$ and $\Pr[\mathrm{G}_0] = \Pr[\, b' = 1 \mid b_{\mathsf{odher}} = 0\,]$, meaning that

$$\mathsf{Adv}^{\mathsf{odher}}_{\mathbb{G},\mathsf{H},\mathcal{O}} = \Pr[\mathrm{G}_1] - \Pr[\mathrm{G}_0]. \tag{5}$$

Compared to the code of $\mathrm{ODHER}^{\mathcal{O}}_{\mathbb{G},\mathsf{H}}$, games $\mathrm{G}_0, \mathrm{G}_1$ replace calls to $\mathsf{H.Ev}$ with explicit calls to the random oracle (according to the construction of H). The code of oracles Ch and RO was also changed. In G_0 procedure Ch always returns elements from map mem, initialized with uniformly random values that are independent of the random oracle. In G_1 oracles Ch and RO now include conditions that ensure consistency between the return values of these functions if they are called with appropriately matching inputs. It follows the behavior we expect when $b = 1$ in game $\mathrm{ODHER}^{\mathcal{O}}_{\mathbb{G},\mathsf{H}}$ because the random oracle would be used to compute the output of Ch in this case.

Note that games G_0 and G_1 are identical until flag bad is set to true. By the fundamental lemma of game playing [6], we have

$$\Pr[\mathrm{G}_1] - \Pr[\mathrm{G}_0] \leq \Pr[\mathsf{bad}^{\mathrm{G}_0}], \tag{6}$$

where $\Pr[\mathsf{bad}^{\mathrm{G}_0}]$ denotes the probability of setting bad to true in game G_0. We will build an adversary $\mathcal{B}$ against the SCDHE security in $\mathbb{G}$ such that

$$\Pr[\mathsf{bad}^{\mathrm{G}_0}] = \Pr[\mathrm{SCDHE}^{\mathcal{B}}_{\mathbb{G}}] = \mathsf{Adv}^{\mathsf{scdhe}}_{\mathbb{G},\mathcal{B}}. \tag{7}$$

Combining Equation (5), Equation (6) and Equation (7) results in $\mathsf{Adv}^{\mathsf{odher}}_{\mathbb{G},\mathsf{H},\mathcal{O}} = \Pr[\mathrm{G}_0] - \Pr[\mathrm{G}_1] \leq \Pr[\mathsf{bad}^{\mathrm{G}_0}] = \mathsf{Adv}^{\mathsf{scdhe}}_{\mathbb{G},\mathcal{B}}$, which confirms the claim in the theorem statement.

The construction of adversary $\mathcal{B}$ is provided in Fig. 20. It simulates game $\mathrm{ODHER}^{\mathcal{O}}_{\mathbb{G},\mathsf{H}}$ for adversary $\mathcal{O}$. The simulation is imperfect, but adversary $\mathcal{O}$ cannot determine that unless it can set bad flag to true when playing in G_0. Adversary $\mathcal{B}$ provides perfect simulation only until this event would happen. Note that bad flag being set true corresponds to $\mathcal{B}$ being able to win in game $\mathrm{SCDHE}^{\mathcal{B}}_{\mathbb{G}}$, so at that point $\mathcal{B}$ can just produce its own output and halt.

In order for the adversary $\mathcal{O}$ to set bad flag true in game G_0, it has to do the following. It has to call $\mathrm{Ch}(s)$ and $\mathrm{RO}(hk' \,\|\, i \,\|\, s \,\|\, e, \mathsf{H.ol})$ such that

$$(hk, v, s, g^{x[v]\cdot y}) = (hk', i, s, e)$$

and such that $\mathrm{Ch}(s) \neq \perp$, where v is the counter in game G_0 when the call to $\mathrm{Ch}(s)$ is made. This means that adversary $\mathcal{O}$ must be able to compute $e = g^{x[v]\cdot y}$ from just $X = g^{x[v]}$ (returned by Up) and $Y = g^y$ (passed as input to adversary). Note that $\mathrm{Ch}(s) \neq \perp$ means that adversary can not call Exp or Hash to trivially deduce this value (neither before calling Up, nor after calling it). The latter also means that adversary must make the call to $\mathrm{RO}(hk' \,\|\, i \,\|\, s \,\|\, e, \mathsf{H.ol})$ itself; it can not implicitly manipulate Hash into making a call to RO with these arguments.

Adversary $\mathcal{B}$ simulates oracles Up, Exp for adversary $\mathcal{O}$ by using its own oracles with the corresponding names. It simulates oracles Ch, Hash, RO for $\mathcal{O}$ by mapping their inputs to uniformly random values on its own. Adversary $\mathcal{B}$ cannot compute the value of $e = g^{x[v]\cdot y}$ to simulate the code of Ch, but it is unique for any choice of (hk, v, s), so $\mathcal{B}$ can map this tuple to a uniformly random value without knowing e. Finally, for every tuple (hk, v, s) processed in Ch and every tuple (hk', i, s, e) processed in RO, adversary will use its DH oracle to check whether $e = g^{x[v]\cdot y}$. If this equality holds, then $\mathcal{B}$ will win in its own game $\mathrm{SCDHE}^{\mathcal{B}}_{\mathbb{G}}$ by returning (v, e).

Adversary $\mathcal{B}^{\mathrm{DH},\textsc{Up},\textsc{Exp}}(g, Y)$

$hk \leftarrow_{\$} \{0,1\}^{\mathsf{H.kl}}$; $v \leftarrow -1$
$\mathcal{O}^{\textsc{UpSim},\textsc{ChSim},\textsc{ExpSim},\textsc{HashSim},\textsc{ROSim}}(hk, g, Y)$
Return (j, Z_j)

$\textsc{UpSim}$

$\mathsf{op} \leftarrow \varepsilon$; $v \leftarrow v + 1$; $X_{cur} \leftarrow \textsc{Up}$; Return X_{cur}

$\textsc{ChSim}(s)$

If (op = "exp") or $((v, s, X_{cur}) \in S_{\mathsf{hash}})$ then
  Return $\perp$
$\mathsf{op} \leftarrow$ "ch" ; $S_{\mathsf{ch}} \leftarrow S_{\mathsf{ch}} \cup \{(v, s, X_{cur})\}$
If $\mathsf{mem}[v, s, X_{cur}] = \perp$ then
  $\mathsf{mem}[v, s, X_{cur}] \leftarrow_{\$} \{0,1\}^{\mathsf{H.ol}}$
For each $(hk', i', s', Z) \in S_{RO}$ do
  If $(hk' \,\|\, i' \,\|\, s' = hk \,\|\, v \,\|\, s)$ and $\mathrm{DH}(X_{cur}, Z)$ then
    $j \leftarrow v$; $Z_j \leftarrow Z$
Return $\mathsf{mem}[v, s, X_{cur}]$

$\textsc{ExpSim}$

If op = "ch" then return $\perp$
$\mathsf{op} \leftarrow$ "exp" ; $x \leftarrow \textsc{Exp}$; Return x

$\textsc{HashSim}(i, s, X)$

If $(i, s, X) \in S_{\mathsf{ch}}$ then return $\perp$
If $i = v$ then $S_{\mathsf{hash}} \leftarrow S_{\mathsf{hash}} \cup \{(i, s, X)\}$
$S_H \leftarrow S_H \cup \{(i, s, X)\}$
If $T_H[i, s, X] = \perp$ then
  $T_H[i, s, X] \leftarrow_{\$} \{0,1\}^{\mathsf{H.ol}}$
  For each $(hk', i', s', Z) \in S_{RO}$ do
    If $(hk' \,\|\, i' \,\|\, s' = hk \,\|\, i \,\|\, s)$ and $\mathrm{DH}(X, Z)$ then
      $z \leftarrow hk \,\|\, i \,\|\, s \,\|\, Z$; $\kappa \leftarrow \mathsf{H.ol}$
      If $T[z, \kappa] \neq \perp$ then $T_H[i, s, X] \leftarrow T[z, \kappa]$
Return $T_H[i, s, X]$

$\textsc{ROSim}(z, \kappa)$

$hk' \,\|\, i \,\|\, s \,\|\, Z \leftarrow z$
$S_{RO} \leftarrow S_{RO} \cup \{(hk', i, s, Z)\}$
If $T[z, \kappa] = \perp$ then
  $T[z, \kappa] \leftarrow_{\$} \{0,1\}^{\kappa}$
  If $\kappa = \mathsf{H.ol}$ then
    For each $(i', s', X) \in S_H$ do
      If $(hk \,\|\, i' \,\|\, s' = hk' \,\|\, i \,\|\, s)$ and $\mathrm{DH}(X, Z)$ then
        $T[z, \kappa] \leftarrow T_H[i, s, X]$
For each $(i', s', X) \in S_{\mathsf{ch}}$ do
  If $(hk \,\|\, i' \,\|\, s' = hk' \,\|\, i \,\|\, s)$ and $\mathrm{DH}(X, Z)$ then
    $j \leftarrow v$; $Z_j \leftarrow Z$
Return $T[z, \kappa]$

Figure 20: Adversary $\mathcal{B}$ for proof of Lemma A.3.

In addition to the above, adversary $\mathcal{B}$ must also maintain consistency between responses from oracles Hash and RO. It does that using its DH oracle in a similar way as above, by checking whether any matching queries (same hk, i, and s) to these oracles satisfy $X^y = Z$. Whenever this

happens, it makes sure that the values returned by these two oracles are consistent; otherwise, it can keep returning uniformly random strings.

Note that for each pair of calls that adversary $\mathcal{O}$ makes to RO and CH (resp. to RO and HASH), adversary $\mathcal{B}$ will make at most one query to its DH oracle (during the later of the two calls). Therefore, adversary $\mathcal{B}$ makes at most $q_{\mathrm{RO}} \cdot (q_{\mathrm{CH}} + q_{\mathrm{HASH}})$ queries to its DH oracle. The running time of adversary $\mathcal{B}$ is roughly that of adversary $\mathcal{O}$, along with the extra computation that is required to evaluate the $q_{\mathrm{RO}} \cdot (q_{\mathrm{CH}} + q_{\mathrm{HASH}})$ conditions as per above. ▮

Reducing SCDHE to SCDH. The second intermediate lemma is as follows.

Lemma A.4 *Let $\mathbb{G}$ be a cyclic group of order $p \in \mathbb{N}$, and let $\mathbb{G}^*$ denote the set of its generators. Let $\mathcal{B}$ be an adversary attacking the* SCDHE*-security of $\mathbb{G}$ that makes q_{DH} queries to its* DH *oracle and q_{UP} queries to its* UP *oracle. Then there is an adversary $\mathcal{S}$ attacking the* SCDH*-security of $\mathbb{G}$ such that*

$$\mathsf{Adv}^{\mathsf{scdhe}}_{\mathbb{G},\mathcal{B}} \leq q_{\mathrm{UP}} \cdot \mathsf{Adv}^{\mathsf{scdh}}_{\mathbb{G},\mathcal{S}}.$$

Adversary $\mathcal{S}$ makes at most q_{DH} queries to its DH *oracle and its running time is approximately that of $\mathcal{B}$.*

Proof of Lemma A.4: We build an adversary $\mathcal{S}$ attacking the SCDH-security in $\mathbb{G}$ as follows:

Adversary $\mathcal{S}^{\mathrm{DH}}(g, X_{\mathcal{S}}, Y)$	UPSIM
$j_{\mathcal{S}} \leftarrow_{\$} \{0, \ldots, q_{\mathrm{UP}} - 1\}$; $v \leftarrow -1$	$v \leftarrow v + 1$; $x \leftarrow_{\$} \mathbb{Z}_p$; $X \leftarrow g^x$
$(j, Z) \leftarrow_{\$} \mathcal{B}^{\mathrm{DHSIM},\mathrm{UPSIM},\mathrm{EXPSIM}}(g, Y)$	If $(v = j_{\mathcal{S}})$ then $X \leftarrow X_{\mathcal{S}}$
Return Z	Return X
DHSIM(X, Z)	EXPSIM
Return $\mathrm{DH}(X, Z)$	If $(v = j_{\mathcal{S}})$ then return $\perp$
	Return x

Adversary $\mathcal{S}$ simulates game $\mathrm{SCDHE}_{\mathbb{G}}$ for adversary $\mathcal{B}$, answering $\mathcal{B}$'s calls to oracle DH using its own DH oracle (it has the same functionality because $\mathcal{S}$ runs $\mathcal{B}$ with g, Y as input). Adversary $\mathcal{S}$ chooses a random index $j_{\mathcal{S}}$ between 0 and $q_{\mathrm{UP}} - 1$, representing its guess of which index j is going to be returned by adversary $\mathcal{B}$. It then uses $X_{\mathcal{S}}$ as the challenge value to answer $\mathcal{B}$'s call to oracle UP when $v = j_{\mathcal{S}}$. It samples its own challenge exponents for the rest of the UP calls, meaning it is also able to answer $\mathcal{B}$'s calls to EXP whenever $v \neq j_{\mathcal{S}}$. Adversary $\mathcal{S}$ is not able to properly simulate the expose oracle for $\mathcal{B}$ when $v = j_{\mathcal{S}}$, but this is not important for our reduction because $\mathcal{B}$ could not have won by returning $j = j_{\mathcal{S}}$ after making such EXP call (it would set $\mathsf{op}[j_{\mathcal{S}}] =$ "exp" in game $\mathrm{SCDHE}_{\mathbb{G}}$).

Let $j_{\mathcal{B}}$ denote the index guessed by $\mathcal{S}$ in game $\mathrm{SCDH}^{\mathcal{S}}_{\mathbb{G}}$, and let $j_{\mathcal{B}}$ denote the index returned by $\mathcal{B}$ in game $\mathrm{SCDHE}^{\mathcal{B}}_{\mathbb{G}}$. For any $j \in \{0, 1, \ldots, q_{\mathrm{UP}} - 1\}$ we have

$$\Pr\left[\, \mathrm{SCDH}^{\mathcal{S}}_{\mathbb{G}} \,|\, j_{\mathcal{S}} = j \,\right] \geq \Pr[\mathrm{SCDHE}^{\mathcal{B}}_{\mathbb{G}} \wedge j_{\mathcal{B}} = j].$$

This is an inequality because it is possible that $\mathcal{S}$ wins in $\mathrm{SCDH}^{\mathcal{S}}_{\mathbb{G}}$ by simulating $\mathcal{B}$ that does not win in $\mathrm{SCDHE}^{\mathcal{B}}_{\mathbb{G}}$ (e.g. this happens if $\mathcal{B}$ returns (j, Z) after calling EXP when $v = j$). We get the following:

$$\mathsf{Adv}^{\mathsf{scdh}}_{\mathbb{G},\mathcal{S}} = \Pr[\mathrm{SCDH}^{\mathcal{S}}_{\mathbb{G}}]$$

$$
\begin{aligned}
&= \sum_{j=0}^{q_{\text{UP}}-1} \Pr[\text{SCDH}_{\mathbb{G}}^{\mathcal{S}} | j_{\mathcal{S}} = j] \cdot \Pr[j_{\mathcal{S}} = j] \\
&\geq \sum_{j=0}^{q_{\text{UP}}-1} \Pr[\text{SCDHE}_{\mathbb{G}}^{\mathcal{B}} \wedge j_{\mathcal{B}} = j] \cdot \frac{1}{q_{\text{UP}}} \\
&= \frac{1}{q_{\text{UP}}} \cdot \Pr[\text{SCDHE}_{\mathbb{G}}^{\mathcal{B}}] \\
&= \frac{1}{q_{\text{UP}}} \cdot \mathsf{Adv}_{\mathbb{G},\mathcal{B}}^{\mathsf{scdhe}}
\end{aligned}
$$

The claimed number of oracle queries and running time of $\mathcal{S}$ follows from its construction. ▮

Proof of Theorem A.1: This theorem is a direct consequence of Lemma A.3 and Lemma A.4.▮

A.2 SCDHE reduction to SCDH with rewinding

In previous section we provided a reduction from ODHE to SCDH in ROM, by combining two intermediate lemmas – Lemma A.3 and Lemma A.4. The latter lemma reduced SCDHE to SCDH and incurred a multiplicative loss in advantage. We now use the self-reducibility property of Diffie-Hellman to state and prove an alternative reduction from SCDHE to SCDH that avoids such multiplicative loss. The new reduction is formalized as Lemma A.5 below. Combining Lemma A.5 with Lemma A.3 from the previous section will yield a more efficient overall reduction from ODHE to SCDH in ROM, as stated in Theorem A.2 of Appendix A.

Outline of the new reduction. Recall that the SCDH adversary $\mathcal{S}$ used for the proof of Lemma A.4 attempts to simulate the provided SCDHE adversary $\mathcal{B}$. Both adversaries take a challenge value $Y = g^y$ as input for some secret uniformly random exponent y. Adversary $\mathcal{S}$ tries to guess which of $\mathcal{B}$'s queries to its oracle UP —returning some group element $X = g^x$— will be used for constructing $\mathcal{B}$'s output value $Z = g^{xy}$. The SCDH adversary then uses the challenge value $X_{\mathcal{S}} = g^{x_{\mathcal{S}}}$ from its own game as the output of the chosen UP query, and answers every other UP query by sampling a uniformly random exponent x itself and returning $X = g^x$.

The new SCDH adversary $\mathcal{S}_u$ for the reduction in this section will instead re-randomize its challenge value $X_{\mathcal{S}_u} = g^{x_{\mathcal{S}_u}}$ in order to answer UP queries for the simulated SCDHE adversary $\mathcal{B}$. To answer an oracle UP query, adversary $\mathcal{S}_u$ will occasionally choose a uniformly random exponent t and return a uniformly random-looking group element $X_{\mathcal{S}_u} \cdot g^t$. If the SCDHE adversary wins its game by returning $Z = (X_{\mathcal{S}_u} \cdot g^t)^y$ then the SCDH adversary can recover $g^{x_{\mathcal{S}_u} y}$ by computing $Z \cdot Y^{-t}$.

However, if adversary $\mathcal{S}_u$ generates its answer to UP query as above, then it cannot answer $\mathcal{B}$'s subsequent query to its EXP oracle, because $\mathcal{S}_u$ does not know the corresponding exponent $x_{\mathcal{S}_u} \cdot t$. To avoid this problem, adversary $\mathcal{S}_u$ for every call to UP guesses whether or not $\mathcal{B}$ will call EXP to recover the corresponding exponent. If adversary $\mathcal{S}_u$ expects a call to EXP, then it answers UP query using $X = g^x$ for a uniformly random exponent x; otherwise, it answers UP query using $X_{\mathcal{S}_u} \cdot g^t$ for a uniformly random exponent t as described above. If $\mathcal{S}_u$ determines that its guess was wrong, it "rewinds" adversary $\mathcal{B}$ back to the corresponding state and attempts to guess again.

$\underline{\text{Game SCDHE}^{\mathcal{B}}_{\mathbb{G}}}$

$g \leftarrow\$ \mathbb{G}^*$; $y \leftarrow\$ \mathbb{Z}_p$; $v \leftarrow -1$

$(\text{proc}, \text{args}, \sigma, \mathsf{done}) \leftarrow\$ \mathcal{B}_1(g, g^y)$

While not done do

  If proc = "dh" then

    $(X, Z) \leftarrow \text{args}$; out $\leftarrow \text{DH}(X, Z)$

  If proc = "up" then out $\leftarrow$ Up

  If proc = "exp" then out $\leftarrow$ Exp

  $(\text{proc}, \text{args}, \sigma, \mathsf{done}) \leftarrow\$ \mathcal{B}_2(\sigma, \text{out})$

$(j, Z) \leftarrow \sigma$

Return $(0 \le j \le v)$ and $(\mathsf{op}[j] \ne \text{"exp"})$ and $(Z = g^{x[j] \cdot y})$

$\underline{\text{DH}(X, Z)}$

Return $(X^y = Z)$

$\underline{\text{Up}}$

$v \leftarrow v + 1$; $x[v] \leftarrow\$ \mathbb{Z}_p$; Return $g^{x[v]}$

$\underline{\text{Exp}}$

$\mathsf{op}[v] = \text{"exp"}$; Return $x[v]$

Figure 21: Game defining SCDHE assumption in group $\mathbb{G}$. This non-black-box definition extends that of Fig. 18 to require that adversary $\mathcal{B} = (\mathcal{B}_1, \mathcal{B}_2)$ halts after every oracle call.

Alternative definition of SCDHE. We now redefine the SCDHE security in group $\mathbb{G}$ in a way that allows to "rewind" the adversary playing in this game. Let $\mathbb{G}$ be a cyclic group of order $p \in \mathbb{N}$, and let $\mathbb{G}^*$ denote the set of its generators. Consider game SCDHE of Fig. 21, associated to group $\mathbb{G}$ and to an adversary $\mathcal{B} = (\mathcal{B}_1, \mathcal{B}_2)$. As opposed to the original definition from Fig. 18, now adversary $\mathcal{B}$ must temporarily halt and output its state whenever it calls one of its oracles. Specifically, adversary $\mathcal{B}$ is required to output a tuple $(\text{proc}, \text{args}, \sigma, \mathsf{done})$ where proc is the name of the oracle it wants to call, args is the argument string it wants to pass to that oracle, σ is the state of $\mathcal{B}$, and done is a boolean flag indicating whether $\mathcal{B}$ has finished playing the game (in which case σ will be interpreted as its final output). The advantage of $\mathcal{B}$ in breaking the SCDHE security of $\mathbb{G}$ is defined as $\mathsf{Adv}^{\mathsf{scdhe}}_{\mathbb{G},\mathcal{B}} = \Pr[\text{SCDHE}^{\mathcal{B}}_{\mathbb{G}}]$. We emphasize that σ represents *all* of the state being stored by $\mathcal{B}$ so "rewinding" $\mathcal{B}$ will simply consist of calling $\mathcal{B}$ again on a prior value of σ.

Reducing SCDHE to SCDH using rewinding. The rewinding-based lemma is as follows.

Lemma A.5 *Let $\mathbb{G}$ be a cyclic group of order $p \in \mathbb{N}$, and let $\mathbb{G}^*$ denote the set of its generators. Let $\mathcal{B}$ be an adversary attacking the* SCDHE*-security of $\mathbb{G}$ that makes q_{DH} queries to its* DH *oracle and q_{Up} queries to its* Up *oracle. Let $u \in \mathbb{N}$. Then there is an adversary $\mathcal{S}_u$ attacking the* SCDH*-security of $\mathbb{G}$ such that*

$$\mathsf{Adv}^{\mathsf{scdhe}}_{\mathbb{G},\mathcal{B}} \le \mathsf{Adv}^{\mathsf{scdh}}_{\mathbb{G},\mathcal{S}_u} + q_{\text{Up}} \cdot 2^{-u}.$$

Let $q^* = 1 + u \cdot q_{\text{Up}}$. Adversary $\mathcal{S}_u$ makes at most $q^* \cdot q_{\text{DH}}$ queries to its DH *oracle. Its running time is approximately q^* times that of $\mathcal{B}$.*

The expected number of oracle queries and running time of adversary $\mathcal{S}_u$ are better than the worst-case guarentees provided above. The expected number of oracle queries to DH made by adversary $\mathcal{S}_u$ is about twice that of $\mathcal{B}$. The expected running time of adversary $\mathcal{S}_u$ is approximately twice

Adversary $\mathcal{S}_u^{\mathrm{DH}}(g, X_{\mathcal{S}_u}, Y)$

$v \leftarrow -1$; $b_g \leftarrow 1$; $(\mathrm{proc}, \mathrm{args}, \sigma, \mathsf{done}) \leftarrow\$ \mathcal{B}_1(g, Y)$
While not done do
  If proc = "dh" then
    $(X, Z) \leftarrow \mathrm{args}$; $\mathrm{out} \leftarrow \mathrm{DH}(X, Z)$
  If proc = "up" then
    If $(b_g = 0)$ and $(\mathsf{op}[v] \neq$ "exp") then Revert
    $\mathsf{maxErr} \leftarrow u$; $\mathsf{lastSave} \leftarrow \sigma$; $v \leftarrow v + 1$
    //label//
    $b_g \leftarrow\$ \{0, 1\}$; $t[v] \leftarrow\$ \mathbb{Z}_p$; $\mathrm{out} \leftarrow X_{\mathcal{S}_u}^{b_g} \cdot g^{t[v]}$
  If proc = "exp" then
    If $(b_g = 1)$ and $(v \neq -1)$ then Revert
    $\mathsf{op}[v] \leftarrow$ "exp" ; $\mathrm{out} \leftarrow t[v]$
  $(\mathrm{proc}, \mathrm{args}, \sigma, \mathsf{done}) \leftarrow\$ \mathcal{B}_2(\sigma, \mathrm{out})$
If $(b_g = 0)$ and $(\mathsf{op}[v] \neq$ "exp") then Revert
$(j, Z) \leftarrow \sigma$; Return $Z \cdot Y^{-t[j]}$

Procedure Revert

$\mathsf{maxErr} \leftarrow \mathsf{maxErr} - 1$
If $\mathsf{maxErr} < 0$ then **abort**
$\sigma \leftarrow \mathsf{lastSave}$
goto label

Figure 22: Adversary $\mathcal{S}_u$ for proof of Lemma A.5.

that of $\mathcal{B}$.

Proof of Lemma A.5: We build an adversary $\mathcal{S}_u$ attacking the SCDH-security in $\mathbb{G}$, as defined in Fig. 22. We use **abort** as a shorthand for $\mathcal{S}_u$ halting execution and returning an output value (j, Z) that guarantess that $\mathcal{S}_u$ loses in game $\mathrm{SCDH}_{\mathbb{G}}^{\mathcal{S}_u}$, for example $(-1, g)$. As a part of adversary $\mathcal{S}_u$, we define an auxiliary procedure Revert that contains the code for rewinding the simulated adversary $\mathcal{B}$. It holds no special meaning in our security proof; its only purpose is to avoid writing duplicate code at every position that calls rewinding, essentially simplifying the definition of $\mathcal{S}_u$.

Adversary $\mathcal{S}_u$ simulates the $\mathrm{SCDHE}_{\mathbb{G}}$ game for adversary $\mathcal{B}$, according to the definition of SCDHE in Fig. 21. Every time adversary $\mathcal{B}$ makes a call to oracle Up, adversary $\mathcal{S}_u$ samples a random bit b_g to determine how to answer this query. If $b_g = 1$ then Up will return $X_{\mathcal{S}_u} \cdot g^{t[v]}$ for a uniformly random exponent $t[v] \in \mathbb{Z}_p$. If $b_g = 0$ then Up will return just $g^{t[v]}$ for a uniformly random exponent $t[v] \in \mathbb{Z}_p$. Furthermore, it is assumed that $\mathcal{B}$ will call oracle Exp prior to making its next query to oracle Up if and only if $b_g = 0$.

The goal here is that adversary $\mathcal{S}_u$ must be capable of answering Exp query whenever this query is made. Note that $\mathcal{S}_u$ can only answer it if $b_g = 0$, unless it can compute the discrete logarithm of $X_{\mathcal{S}_u}$ in group $\mathbb{G}$. On the other hand, for any challenge value returned by Up that was not exposed, adversary $\mathcal{S}_u$ must ensure that it was generated as $X_{\mathcal{S}_u} \cdot g^{t[v]}$ for a known value $t[v]$. This guarantees that if $\mathcal{B}$ wins in game $\mathrm{SCDHE}_{\mathbb{G}}^{\mathcal{B}}$ by returning some (j, Z) such that $Z = X_{\mathcal{S}_u}^{y} \cdot g^{t[j] \cdot y}$, then adversary $\mathcal{S}_u$ will always be able to compute $Z \cdot Y^{-t[j]} = X_{\mathcal{S}_u}^{y}$ to win in game $\mathrm{SCDH}_{\mathbb{G}}^{\mathcal{S}_u}$.

Every time adversary $\mathcal{B}$ makes a call to oracle Up, adversary $\mathcal{S}_u$ checks whether its previous guess about $\mathcal{B}$'s behavior (as reflected by b_g) was correct. If the guess was correct, then $\mathcal{S}_u$ saves $\mathcal{B}$'s current state σ in variable lastSave and proceeds to make a new guess about $\mathcal{B}$'s future behavior. If the guess was wrong, then $\mathcal{S}_u$ "rewinds" adversary $\mathcal{B}$ back to the state it had during its previous call to oracle Up, by restoring $\mathcal{B}$'s state from the current value of lastSave and calling **goto label** to move the instruction pointer to **//label//**; adversary $\mathcal{S}_u$ then makes a new attempt to guess $\mathcal{B}$'s behavior, sampling a random bit b_g. We ensure that $\mathcal{S}_u$ eventually halts by allowing it to make at most maxErr wrong guesses in a row. If $\mathcal{S}_u$ exceeds this number of wrong guesses, it immediately

```
Games G0–G1
g ←$ G* ; y ←$ Zp ; x ←$ Zp ; v ← −1 ; b_g ← 1
(proc, args, σ, done) ←$ B1(g, g^y)
While not done do
  If proc = "dh" then
    (X, Z) ← args ; out ← (X^y = Z)
  If proc = "up" then
    If (b_g = 0) and (op[v] ≠ "exp") then REVERT
    maxErr ← u ; lastSave ← σ ; v ← v + 1
    //label//
    b_g ←$ {0,1} ; t[v] ←$ Zp ; out ← g^(x·b_g) · g^t[v]
  If proc = "exp" then
    If (b_g = 1) and (v ≠ −1) then REVERT
    op[v] ← "exp" ; out ← t[v]
  (proc, args, σ, done) ←$ B2(σ, out)
If (b_g = 0) and (op[v] ≠ "exp") then REVERT
(j, Z) ← σ
Return (Z · g^(−t[j]·y) = g^xy)

Procedure REVERT
maxErr ← maxErr − 1
If maxErr < 0 then
  bad_v ← true
  abort                                          // G0
(v, σ) ← lastSave
goto label
```

Figure 23: Games $\mathrm{G}_0, \mathrm{G}_1$ for proof of Lemma A.5.

calls **abort** to halt its execution.

We now analyze the success probability of adversary $\mathcal{S}_u$. Consider games $\mathrm{G}_0, \mathrm{G}_1$ of Fig. 23. Lines not annotated with comments are common to both games. Game G_0 is equivalent to $\mathrm{SCDH}_{\mathbb{G}}^{\mathcal{S}_u}$, with the code of $\mathcal{S}$ inserted, and simplified to reference g^x instead of $X_{\mathcal{S}_u}$. It follows that

$$\Pr[\mathrm{G}_0] = \Pr[\mathrm{SCDH}_{\mathbb{G}}^{\mathcal{S}_u}]. \tag{8}$$

Game G_1 is equivalent to G_0, except that it never calls **abort**. Let $\Pr[\mathsf{bad}^{\mathrm{G}_0}]$ denote the probability that a bad_v flag is set true in game G_0 for some $v \in \{0, 1, \ldots, q_{\mathrm{UP}} - 1\}$. Note that **abort** is only called when this happens, and the two games are identical-until-bad. According to the fundamental lemma of game playing [6], we have

$$\Pr[\mathrm{G}_1] \leq \Pr[\mathrm{G}_0] + \Pr[\mathsf{bad}^{\mathrm{G}_0}]. \tag{9}$$

Note that adversary $\mathcal{B}$ always gets uniformly random group elements in response to its oracle UP queries, regardless of the value of b_g. Meaning that regardless of the behavior of $\mathcal{B}$ the probability that b_g is chosen correctly at any point is exactly 1/2. It follows that the output distribution of $\mathcal{B}$ induced in $\Pr[\mathrm{G}_1]$ is exactly that same as that induced in $\mathrm{SCDHE}_{\mathbb{G}}^{\mathcal{B}}$. It follows that the condition $Z \cdot g^{-t[j]\cdot y} = g^{xy}$ in game $\Pr[\mathrm{G}_1]$ will hold whenever (j, Z) would have been a correct answer in game SCDHE. This gives us

$$\Pr[\mathrm{SCDHE}_{\mathbb{G}}^{\mathcal{B}}] \leq \Pr[\mathrm{G}_1]. \tag{10}$$

Game $\text{UFORGE}^{\mathcal{M}}_{\mathsf{RKE}}$

$(k, sek_s, (stk_s, stk_r, sek_r)) \leftarrow\$ \mathsf{RKE.IKg}$
$k_s \leftarrow k\ ;\ k_r \leftarrow k\ ;\ upd \leftarrow\$ \mathcal{M}^{\text{Ratchet}}(stk_s)$
$(sek_r, k_r, acc) \leftarrow\$ \mathsf{RKE.RKg}(stk_s, stk_r, sek_r, upd, k_r)$
Return acc

Ratchet

$prev \leftarrow (r, sek_s, k_s)\ ;\ r \leftarrow\$ \mathsf{RKE.RS}$
$(sek_s, k_s, upd) \leftarrow \mathsf{RKE.SKg}(stk_s, sek_s; r)$
$(sek_r, k_r, acc) \leftarrow\$ \mathsf{RKE.RKg}(stk_s, stk_r, sek_r, upd, k_r)$
Return $(prev, upd)$

Figure 24: Game update unforgeability of ratcheted key exchange scheme RKE.

Finally, we now bound the probability that a bad flag is set in G_0 by the following inequalities.

$$\Pr[\mathsf{bad}^{\mathrm{G}_0}] \leq \sum_{v=0}^{q_{\text{UP}}-1} \Pr[\mathsf{bad}_v^{\mathrm{G}_0}] \leq \sum_{v=0}^{q_{\text{UP}}-1} 2^{-u} = q_{\text{UP}} \cdot 2^{-u}. \tag{11}$$

The first inequality comes from a simple union bound. To see the second inequality note that for any particular bad_v to be set true the guess bit b_g must be incorrect u times in a row, which happens with probability 2^{-u}.

Finally, the theorem statement follows from Equations (8)–(11):

$$\Pr[\text{SCDHE}^{\mathcal{B}}_{\mathbb{G}}] \leq \Pr[\text{SCDH}^{\mathcal{S}_u}_{\mathbb{G}}] + q_{\text{UP}} \cdot 2^{-u}.$$

The claimed number of oracle queries and the worst-case running time of $\mathcal{S}_u$ follows from the fact that $\mathcal{S}_u$ will run $\mathcal{B}$ once, and depending on $\mathcal{B}$'s behavior it will rewind $\mathcal{B}$ at most $u \cdot q_{\text{UP}}$ times. In the worst case, each rewinding will reset $\mathcal{B}$ back to the very beginning of its execution and it would have to be run all over again. ▮

Proof of Theorem A.2: This theorem is a direct consequence of Lemma A.3 and Lemma A.5.▮

B Necessity of authenticaticating the update information

In this section we show that if an attacker can forge update information *upd* for a ratcheted key exchange scheme, then it can be used to break the KIND security of this scheme. Here, a *forged* update information is one that was not produced by the sender, but would still be accepted by the receiver. This result could equivalently be stated as reducing the UFORGE security (that we are about to formally define) of a ratcheted key exchange scheme to its KIND security, meaning that UFORGE is a property that is a necessary condition for KIND security.

Update unforgeability. Informally, a ratcheted key exchange scheme is secure against update forgeries if an adversary, given access to several samples of update information and to all sender's secrets prior to the generation of these samples, cannot generate its own update information that will be accepted by the receiver. Conversely, an adversary is good at update forgeries if it can do this.

Consider game UFORGE of Fig. 24, associated to a ratcheted key exchange scheme RKE and an adversary $\mathcal{M}$. The advantage of $\mathcal{M}$ at breaking the UFORGE security of RKE is given by

$\mathsf{Adv}^{\mathsf{uforge}}_{\mathsf{RKE},\mathcal{M}} = \Pr[\mathrm{UFORGE}^{\mathcal{M}}_{\mathsf{RKE}}]$.

Adversary $\mathcal{M}$ is given the sender's static key and is provided with an access to an oracle Ratchet that performs a ratcheted key update for both the sender and the receiver. Oracle Ratchet returns the sender's secrets from before ratcheting the keys, along with the update information *upd* that was used for ratcheting receiver's key. Adversary $\mathcal{M}$ can call its oracle an arbitrary number of times, and its goal is to eventually generate its own update information *upd* that is accepted by the receiver. Note that the first call to oracle Ratchet returns (r, sek_s, k_s) where r is not explicitly initialized; according to our notation conventions from Section 2, it is implicitly assumed to be initialized with 0.

Breaking KIND using an update forgery. We now show that any adversary $\mathcal{M}$ that is successful at attacking the UFORGE security of a ratcheted key exchange scheme RKE can be used to build an adversary $\mathcal{D}$ that is successful at attacking the KIND security of RKE.

Theorem B.1 *Let* RKE *be a ratcheted key exchange scheme. Let $\mathcal{M}$ be an adversary attacking the* UFORGE*-security of* RKE *that makes q_{Ratchet} queries to its* Ratchet *oracle. Then there is an adversary $\mathcal{D}$ attacking the* KIND*-security of* RKE *such that*

$$\mathsf{Adv}^{\mathsf{kind}}_{\mathsf{RKE},\mathcal{D}} = \left(\frac{1}{2} - \frac{1}{2^{\mathsf{RKE.kl}+1}}\right) \cdot \mathsf{Adv}^{\mathsf{uforge}}_{\mathsf{RKE},\mathcal{M}}.$$

Adversary $\mathcal{D}$ makes at most $q_{\mathrm{Ratchet}} + 1$ queries to each of its RatSend*,* RatRec*, and* Exp *oracles. It makes one query to each of its* ChSend *and* ChRec *oracles. Its running time is approximately that of $\mathcal{M}$.*

The starting idea for the KIND adversary $\mathcal{D}$ is to cause the receiver to update its keys with update information *upd* that was not generated by the sender. It does that by calling its oracle RatRec on update information *upd* that it learns from UFORGE adversary $\mathcal{M}$. Then (after another call to RatSend) oracles ChSend and ChRec will presumably return differing keys in the "real world" ($b = 1$) and matching keys in the "random world" ($b = 0$). Comparing these keys gives a straightforward attack. We note that this attack does not require $\mathcal{D}$ to call Exp right before calling oracles RatSend, ChSend, and ChRec. Hence the restricted flag is false at the time when these challenge oracles are called.

Unfortunately we cannot prove that this attack —as outlined above— is successful for an arbitrary RKE scheme. The difficulty stems from the fact that RKE will not necessarily generate non-matching values in the "real world". For "natural" ratcheted key exchange schemes it seems highly unlikely that keys generated by the two challenge oracles in the "real world" would match. However, rigorously proving this claim is hard.

We will augment the attack idea described above to call Exp with probability 1/2 prior to calling oracles RatSend, ChSend, and ChRec. When Exp is called prior to this sequence of calls, game $\mathrm{KIND}^{\mathcal{D}}_{\mathsf{RKE}}$ sets restricted to true. Depending on whether Exp is called, adversary $\mathcal{D}$ is also defined to flip its output bit.

In the "real world" ($b = 1$) of game $\mathrm{KIND}^{\mathcal{D}}_{\mathsf{RKE}}$, the output of the challenge oracles is the same regardless of whether restricted is true. So when $b = 1$ the interaction of $\mathcal{D}$ with its challenge oracles in game $\mathrm{KIND}^{\mathcal{D}}_{\mathsf{RKE}}$ does not change depending on whether Exp was called (which happens with probability 1/2), but its output bit is nonetheless flipped. In this case adversary $\mathcal{D}$ will succeed to guess the challenge bit (meaning it will return $b' = 1$) with probability 1/2.

In constract, calling Exp in the "random world" ($b = 0$) of game $\mathrm{KIND}^{\mathcal{D}}_{\mathsf{RKE}}$ results in the challenge oracles returning outputs that have different equality patterns. If Exp was called, then

restricted is true and adversary will get different keys with high probability in the "random world", because ChSend returns a uniformly random key and ChRec returns a real key. If Exp was not called, then restricted is false and adversary will always get same keys from the challenge oracles in the "random world". By flipping adversary's output value in one of these two cases, we get an adversary that guesses the correct challenge bit (meaning it returns $b' = 0$) with probability close to 1. This is sufficient to distinguish between the "real world" and the "random world".

Proof of Theorem B.1: We build an adversary $\mathcal{D}$ attacking the KIND-security of RKE as follows:

Adversary $\mathcal{D}^{\text{RatSend},\text{RatRec},\text{Exp},\text{ChSend},\text{ChRec}}(stk_s)$	RatchetSim
$upd \leftarrow_{\$} \mathcal{M}^{\text{RatchetSim}}(stk_s)$; $b_{\mathcal{D}} \leftarrow_{\$} \{0,1\}$	$prev \leftarrow$ Exp
If $(b_{\mathcal{D}} = 1)$ then Exp	$upd \leftarrow$ RatSend
$acc_{\mathcal{D}} \leftarrow$ RatRec(upd)	RatRec(upd)
If not $acc_{\mathcal{D}}$ then return 1	Return $(prev, upd)$
RatSend ; $k_s \leftarrow$ ChSend ; $k_r \leftarrow$ ChRec	
If $(k_s = k_r)$ then return $b_{\mathcal{D}}$ else return $1 - b_{\mathcal{D}}$	

Adversary $\mathcal{D}$ first uses adversary $\mathcal{M}$ to generate forged update information upd (the simulation of $\mathcal{M}$'s oracle RatchetSim is straightforward). Then $\mathcal{D}$ samples a bit $b_{\mathcal{D}}$ to decide whether it should make a call to Exp before proceeding. It then calls RatRec with update information upd to derive key k_r (immediately returning 1 if this update fails), calls RatSend to derive what would have been the "corresponding" key k_s, and finally calls both challenge oracles to get k_s and k_r. If $k_s = k_r$ then $\mathcal{D}$ returns $b_{\mathcal{D}}$ as its output, otherwise it returns $1 - b_{\mathcal{D}}$.

Consider game $\text{KIND}^{\mathcal{D}}_{\mathsf{RKE}}$. Let b denote the challenge bit in this game. Let $b_{\mathcal{D}}$ and $acc_{\mathcal{D}}$ be the values computed by adversary $\mathcal{D}$ playing in this game. Then the following will hold:

$$\Pr[\,\text{KIND}^{\mathcal{D}}_{\mathsf{RKE}} \,|\, b = 1, b_{\mathcal{D}} = 1, acc_{\mathcal{D}}\,] = 1 - \Pr[\,\text{KIND}^{\mathcal{D}}_{\mathsf{RKE}} \,|\, b = 1, b_{\mathcal{D}} = 0, acc_{\mathcal{D}}\,], \tag{12}$$

$$\Pr[\,\text{KIND}^{\mathcal{D}}_{\mathsf{RKE}} \,|\, b = 0, b_{\mathcal{D}} = 1, acc_{\mathcal{D}}\,] = 1 - 2^{-\mathsf{RKE.kl}}, \tag{13}$$

$$\Pr[\,\text{KIND}^{\mathcal{D}}_{\mathsf{RKE}} \,|\, b = 0, b_{\mathcal{D}} = 0, acc_{\mathcal{D}}\,] = 1. \tag{14}$$

We now justify each of these equations. For this purpose, let k_s and k_r be the values computed by adversary $\mathcal{D}$ playing in game $\text{KIND}^{\mathcal{D}}_{\mathsf{RKE}}$.

To justify Equation (12), note that if $b = 1$ then the values of k_s and k_r that are compared by $\mathcal{D}$ will both be generated by RKE in game $\text{KIND}^{\mathcal{D}}_{\mathsf{RKE}}$, regardless of whether Exp has been called. Thus the probability of k_s and k_r being equal does not depend on the bit $b_{\mathcal{D}}$. Assume that $b = 1$ and $acc_{\mathcal{D}} = \mathsf{true}$. If $b_{\mathcal{D}} = 1$ then $\mathcal{D}$ returns 1 (the correct value of b) whenever $k_s = k_r$. If $b_{\mathcal{D}} = 0$ then $\mathcal{D}$ returns 1 whenever $k_s \neq k_r$. This implies the stated equality.

When $b = 0$ and $b_{\mathcal{D}} = 1$, the value of k_s will have been chosen uniformly at random, but k_r will have been generated by RKE because the restricted flag will be set to true. Thus the probability of them being equal is $2^{-\mathsf{RKE.kl}}$. Assume that $b = 0$ and $acc_{\mathcal{D}} = \mathsf{true}$. If $b_{\mathcal{D}} = 1$ then $\mathcal{D}$ returns 0 (the correct value of b) whenever $k_s \neq k_r$. This implies Equation (13).

When $b = 0$ and $b_{\mathcal{D}} = 0$, the restricted flag will be false, so game $\text{KIND}^{\mathcal{D}}_{\mathsf{RKE}}$ forces k_s and k_r to be the same random value. Assume that $b = 0$ and $acc_{\mathcal{D}} = \mathsf{true}$. If $b_{\mathcal{D}} = 0$ then $\mathcal{D}$ returns 0 (the correct value of b) whenever $k_s = k_r$. This implies Equation (14).

We now use Equations (12)–(14) to compute the conditional probability that $\mathcal{D}$ wins in game $\mathrm{KIND}^{\mathcal{D}}_{\mathsf{RKE}}$, given that $acc_{\mathcal{D}} = \mathsf{true}$. The result is as follows:

$$\begin{aligned}
\Pr[\,\mathrm{KIND}^{\mathcal{D}}_{\mathsf{RKE}} \,|\, acc_{\mathcal{D}}\,] &= \sum_{c\in\{0,1\}} \sum_{d\in\{0,1\}} \Pr[\,\mathrm{KIND}^{\mathcal{D}}_{\mathsf{RKE}} \,|\, b = c, b_{\mathcal{D}} = d, acc_{\mathcal{D}}\,] \cdot \frac{1}{4} \\
&= \frac{1}{4} \cdot \left(1 + \left(1 - 2^{-\mathsf{RKE.kl}}\right) + 1\right) \\
&= \frac{1}{4} \cdot \left(3 - 2^{-\mathsf{RKE.kl}}\right). \qquad (15)
\end{aligned}$$

Note that adversary $\mathcal{D}$ setting $acc_{\mathcal{D}} = \mathsf{true}$ while playing in game $\mathrm{KIND}^{\mathcal{D}}_{\mathsf{RKE}}$ is equivalent to adversary $\mathcal{M}$ winning in game $\mathrm{UFORGE}^{\mathcal{M}}_{\mathsf{RKE}}$. It follows that

$$\Pr[acc_{\mathcal{D}}] = \mathsf{Adv}^{\mathsf{uforge}}_{\mathsf{RKE},\mathcal{M}}. \qquad (16)$$

Furthermore, if $acc_{\mathcal{D}} = \mathsf{false}$ then adversary $\mathcal{D}$ wins in game $\mathrm{KIND}^{\mathcal{D}}_{\mathsf{RKE}}$ with probability 1/2 because (prior to halting) it only queries oracles RatSend, RatRec, Exp that behave independently of the challenge bit in game $\mathrm{KIND}^{\mathcal{D}}_{\mathsf{RKE}}$. We have

$$\Pr[\,\mathrm{KIND}^{\mathcal{D}}_{\mathsf{RKE}} \,|\, \neg acc_{\mathcal{D}}\,] = \frac{1}{2}. \qquad (17)$$

Finally, we compute the advantage of adversary $\mathcal{D}$ from Equations (15)–(17) as follows:

$$\begin{aligned}
\mathsf{Adv}^{\mathsf{kind}}_{\mathsf{RKE},\mathcal{D}} &= 2 \cdot \Pr[\mathrm{KIND}^{\mathcal{D}}_{\mathsf{RKE}}] - 1 \\
&= 2 \cdot \Pr[\,\mathrm{KIND}^{\mathcal{D}}_{\mathsf{RKE}} \,|\, acc_{\mathcal{D}}\,] \cdot \Pr[acc_{\mathcal{D}}] \\
&\quad + 2 \cdot \Pr[\,\mathrm{KIND}^{\mathcal{D}}_{\mathsf{RKE}} \,|\, \neg acc_{\mathcal{D}}\,] \cdot \Pr[\neg acc_{\mathcal{D}}] - 1 \\
&= 2 \cdot \frac{1}{4} \cdot \left(3 - 2^{-\mathsf{RKE.kl}}\right) \cdot \mathsf{Adv}^{\mathsf{uforge}}_{\mathsf{RKE},\mathcal{M}} + 2 \cdot \frac{1}{2} \cdot (1 - \mathsf{Adv}^{\mathsf{uforge}}_{\mathsf{RKE},\mathcal{M}}) - 1 \\
&= \left(\frac{1}{2} - \frac{1}{2^{\mathsf{RKE.kl}+1}}\right) \cdot \mathsf{Adv}^{\mathsf{uforge}}_{\mathsf{RKE},\mathcal{M}}.
\end{aligned}$$

The number of oracle queries and the running time that were claimed in the theorem statement follow from the construction of adversary $\mathcal{D}$. ∎