Ratcheted Encryption and Key Exchange: The Security of Messaging

Mihir Bellare[1], Asha Camper Singh[2], Joseph Jaeger[1], Maya Nyayapati[2], and Igors Stepanovs[1]

[1] Department of Computer Science and Engineering, University of California San Diego, USA. {mihir,jsjaeger,istepano}@eng.ucsd.edu
[2] Salesforce.

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 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) [10] 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 [19].) Ratcheting was then used by Open Whisper Systems in their Signal protocol [22], 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 [25] 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"* [25, 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 [10] it works roughly like this:

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

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, unexposed 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 [23], 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 [8]. 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 [10] 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 [23], 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 at-

tack 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 [10] 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 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 [21]). We show this in [7]. 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 2 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 [8], but a hybrid argument reduces this to the standard single-user security defined in [23]. 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 [9]. 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) [11] 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 [18] 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 [21] assumption while we prove it under the weaker SCDH [1] assumption. Ultimately their work and ours have somewhat different goals. Theirs is to ana-

lyze 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) [12] 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 [13–15] 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 $1 \le i \le j \le |x|$. 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 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. 2 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

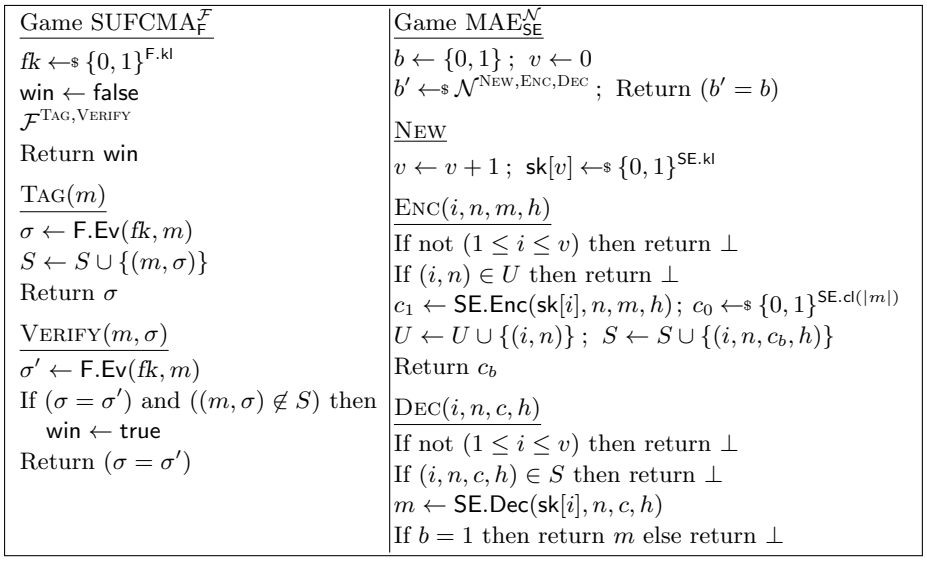

Fig. 1. Games defining strong unforgeability of function family F under chosen message attack, and multi-user authenticated encryption security of SE.

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 TAG oracle with m_{forge} as input. The advantage of $\mathcal{F}$ in breaking the SUFCMA security of F is defined as $\mathsf{Adv}^{\mathsf{sufcma}}_{\mathsf{F},\mathcal{F}} = \Pr[\text{SUFCMA}^{\mathcal{F}}_{\mathsf{F}}]$. 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 algorithms SE.Enc and SE.Dec. Associated to SE is a key length $\mathsf{SE.kl} \in \mathbb{N}$, a nonce space SE.NS, and a ciphertext length function $\mathsf{SE.cl}\colon \mathbb{N} \to \mathbb{N}$. Encryption algorithm 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 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 [24], whereas [23] 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 [23] to the multi-user setting, first formalized in [8]. 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 $\mathsf{H.Ev}(hk, i \,\|\, s \,\|\, X^y)$. 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

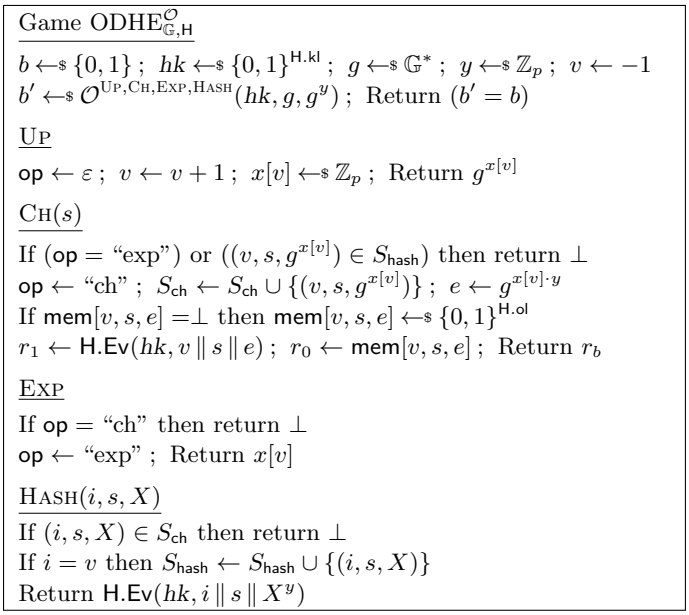

Fig. 2. Game defining Oracle Diffie-Hellman with Exposures assumption for $\mathbb{G}, \mathsf{H}$.

(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 $\mathsf{Adv}^{\mathsf{odhe}}_{\mathbb{G},\mathsf{H},\mathcal{O}} = 2\Pr[\text{ODHE}^{\mathcal{O}}_{\mathbb{G},\mathsf{H}}] - 1$.

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 [21]. In [7] 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 agree-

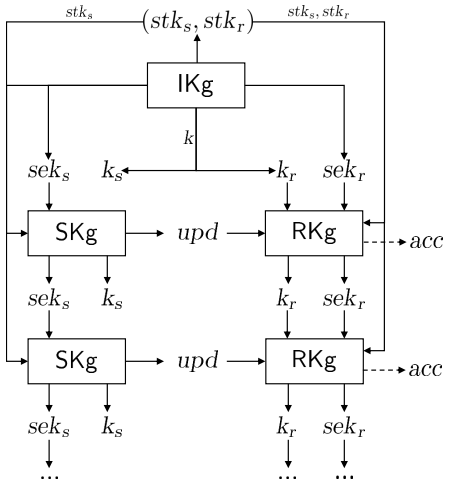

Fig. 3. The interaction between ratcheted key exchange algorithms.

ment 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 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 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 algorithm $\mathsf{RKE.SKg}$ 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 $\mathsf{RKE.RKg}$ 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 $\mathsf{R.SKg}$ to generate a new sender's output key k_s along with the corresponding update information upd; it then runs $\mathsf{R.RKg}$ 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.

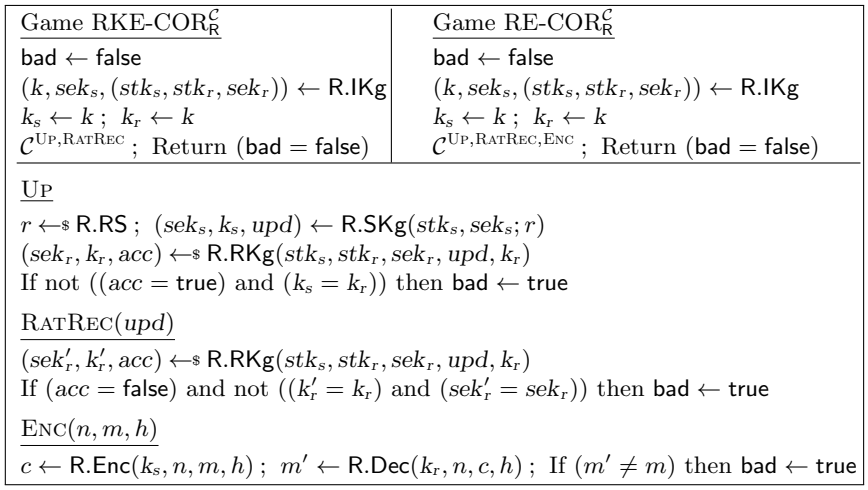

Fig. 4. Game RKE-COR defining correctness of ratcheted key exchange scheme R, and game RE-COR defining correctness of ratcheted encryption scheme R. Oracles Up and RatRec are used in both games, whereas oracle Enc is only used in game RE-COR.

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 [7].

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

Game $\text{KIND}^{\mathcal{D}}_{\mathsf{RKE}}$	Game $\text{RAE}^{\mathcal{A}}_{\mathsf{RE}}$
$b \leftarrow\$ \{0,1\}$; $i_s \leftarrow 0$; $i_r \leftarrow 0$ $(k, sek_s, (stk_s, stk_r, sek_r)) \leftarrow\$ \mathsf{RKE.IKg}$ $k_s \leftarrow k$; $k_r \leftarrow k$ $b' \leftarrow\$ \mathcal{D}^{\text{RatSend},\text{RatRec},\text{Exp},\text{ChSend},\text{ChRec}}(stk_s)$ Return $(b' = b)$	$b \leftarrow\$ \{0,1\}$; $i_s \leftarrow 0$; $i_r \leftarrow 0$ $(k, sek_s, (stk_s, stk_r, sek_r)) \leftarrow\$ \mathsf{RE.IKg}$ $k_s \leftarrow k$; $k_r \leftarrow k$ $b' \leftarrow\$ \mathcal{A}^{\text{RatSend},\text{RatRec},\text{Exp},\text{Enc},\text{Dec}}(stk_s)$ Return $(b' = b)$
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	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
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	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
Exp If $\mathsf{op}[i_s] =$ "ch" then return $\perp$ $\mathsf{op}[i_s] \leftarrow$ "exp" ; Return (r, sek_s, k_s)	Exp If $\mathsf{op}[i_s] =$ "ch" then return $\perp$ $\mathsf{op}[i_s] \leftarrow$ "exp" ; Return (r, sek_s, k_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 else return $\mathsf{rkey}[i_s]$	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
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 $\mathsf{rkey}[i_r]$	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 $\perp$

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

sender (i.e. $upd = \mathsf{auth}[i_r]$) to RatRec, if upd is accepted by the receiver then the $\mathsf{restricted}$ 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.

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 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 RATCHET-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

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 $((i_r+1, fk_r), k_r, acc)$

Fig. 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

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

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 $\textsc{RatRec}$ without having called $\textsc{Exp}$ 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 $\textsc{RatSend}, \textsc{RatRec}, \textsc{Exp}, \textsc{ChSend}, \textsc{ChRec}$, 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 [7] 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 [20, 16] — which is used in the Signal protocol [22] — 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, [16] 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 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.

Some explanation has been removed from the proof below due to lack of space. A more detailed proof is available in the full version of the paper [7].

Proof (Theorem 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 $\mathrm{G}_{0,0}, \ldots, \mathrm{G}_{0,q_{\text{RatSend}}+1}$. 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 $j \in \{0, \ldots, q_{\text{RatSend}}\}$,

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

Furthermore, for all $j \in \{1, \ldots, q_{\text{RatSend}}\}$, 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).

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"
$x \leftarrow\$ \mathbb{Z}_p$; $\sigma \leftarrow \mathsf{F.Ev}(fk_s, g^x)$; $upd \leftarrow (g^x, \sigma)$
$s \leftarrow \mathsf{H.Ev}(hk, i_s \,\|\, \sigma \,\|\, g^x \,\|\, g^{xy})$
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
  $\mathsf{unchanged} \leftarrow \mathsf{false}$
$s \leftarrow \mathsf{H.Ev}(hk, i_r \,\|\, \sigma \,\|\, X \,\|\, X^y)$
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 $\mathsf{rkey}[i_r]$

Fig. 8. Games $\mathrm{G}_{0,j}, \mathrm{G}^*_{0,j}, \mathrm{I}_j$ for proof of Theorem 1.

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 $\mathsf{op}[i_s] = \bot$ then $\mathsf{op}[i_s] \leftarrow$ "ch"
$x \leftarrow\$ \mathbb{Z}_p$; $\sigma \leftarrow \mathsf{F.Ev}(fk_s[i_s], g^x)$; $upd \leftarrow (g^x, \sigma)$
$s \leftarrow \mathsf{H.Ev}(hk, i_s \,\|\, \sigma \,\|\, g^x \,\|\, g^{xy})$ // 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
  $\mathsf{unchanged} \leftarrow \mathsf{false}$
  $s \leftarrow \mathsf{H.Ev}(hk, i_r \,\|\, \sigma \,\|\, X \,\|\, X^y)$
  $i_r \leftarrow i_r + 1$; $k_r \leftarrow s[1 \ldots \mathsf{RKE.kl}]$
  $fk_r \leftarrow s[\mathsf{RKE.kl} + 1 \ldots \mathsf{RKE.kl} + \mathsf{F.kl}]$
Return true

Exp

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

ChSend

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

Fig. 9. Games $\mathrm{G}_1, \mathrm{G}_2$ for proof of Theorem 1.

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 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 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 $j \in \{1, \ldots, q_{\mathrm{RatSend}}\}$, 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

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

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.

To complete the hybrid argument part of the proof, we bound the probability that bad gets set true in I_j. Adversary $\mathcal{F}$ (shown in Fig. 11) guesses when $\mathcal{D}$ will first create a forgery and uses that to create its own forgery. Thus for $j \in \{0, \ldots, q_{\mathrm{RatSend}}\}$, $\Pr[\mathsf{bad}^{\mathrm{I}_j}] \leq \Pr[\, \mathrm{SUFCMA}^{\mathcal{F}}_{\mathsf{F}} \,|\, j' = j \,]$ which gives $\mathsf{Adv}^{\mathsf{sufcma}}_{\mathsf{F},\mathcal{F}} \geq (1/(q_{\mathrm{RatRec}} + 1)) \sum_{j=0}^{q_{\mathrm{RatRec}}} \Pr[\mathsf{bad}^{\mathrm{I}_j}]$.

The above work allows us to transition to game $\mathrm{G}_{0,q_{\mathrm{RatSend}}+1}$ 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 $j = q_{\mathrm{RatSend}}$.

$$\begin{aligned}
&\Pr[\mathrm{KIND}^{\mathcal{D}}_{\mathsf{RKE}}] = \Pr[\mathrm{G}_{0,0}] = \Pr[\mathrm{G}_{1,q_{\mathrm{RatSend}}}] + \textstyle\sum_j \Pr[\mathrm{G}_{0,j}] - \Pr[\mathrm{G}^*_{0,j}] \\
&\leq \Pr[\mathrm{G}_{1,q_{\mathrm{RatSend}}}] + \textstyle\sum_j \Pr[\mathsf{bad}^{\mathrm{G}^*_{0,j}}] \\
&= \Pr[\mathrm{G}_{1,q_{\mathrm{RatSend}}}] + q_{\mathrm{RatSend}} \cdot \mathsf{Adv}^{\mathsf{odhe}}_{\mathbb{G},\mathsf{H},\mathcal{O}_1} + \textstyle\sum_j \Pr[\mathsf{bad}^{\mathrm{I}_j}]
\end{aligned}$$

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

RatSendSim
If $\mathsf{op}[i_s] = \perp$ then $\mathsf{op}[i_s] \leftarrow$ “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 $\mathsf{op}[i_s] =$ “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 $\mathsf{rkey}[i_s]$

ChRecSim
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 $\mathsf{rkey}[i_r]$

Fig. 10. Adversary $\mathcal{O}_1$ for proof of Theorem 1.

$$\leq q_{\text{RatSend}} \cdot \mathsf{Adv}^{\mathsf{odhe}}_{\mathbb{G},\mathsf{H},\mathcal{O}_1} + (q_{\text{RatSend}} + 1) \cdot \mathsf{Adv}^{\mathsf{sufcma}}_{\mathsf{F},\mathcal{F}} + \Pr[\mathrm{G}_{1,q_{\text{RatSend}}}].$$

Game G_1 is identical to $\mathrm{G}_{0,q_{\text{RatSend}}+1}$, 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 RatRec has been simplified because $i_r < q_{\text{RatSend}}+1$ always holds when $\mathsf{unchanged}$ is true. Additionally, when $\mathsf{unchanged}$ is true (and thus upd has been directly forwarded between RatSend and 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 $\Pr[\mathrm{G}_{0,q_{\text{RatSend}}+1}] = \Pr[\mathrm{G}_1]$.

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 Exp is called in which case we reset them to the correct output of H). We bound the difference between $\Pr[\mathrm{G}_1]$

Adversary $\mathcal{F}^{\text{TAG,VERIFY}}$

$j' \leftarrow\$ \{0,\ldots,q_{\text{RATSEND}}\}$; $b \leftarrow\$ \{0,1\}$
$i_s \leftarrow 0$; $i_r \leftarrow 0$; $\mathsf{unchanged} \leftarrow \mathsf{true}$
$rand \leftarrow\$ \{0,1\}^{\mathsf{H.ol}}$; $k_s \leftarrow\$ \{0,1\}^{\mathsf{RKE.kl}}$; $k_r \leftarrow k_s$
$fk_s \leftarrow\$ \{0,1\}^{\mathsf{F.kl}}$; $fk_r \leftarrow fk_s$; $hk \leftarrow\$ \{0,1\}^{\mathsf{H.kl}}$
$g \leftarrow\$ \mathbb{G}^*$; $y \leftarrow\$ \mathbb{Z}_p$; $stk_s \leftarrow (hk, g, g^y)$
$\mathcal{D}^{\text{RATSENDSIM,RATRECSIM,EXPSIM,CHSENDSIM,CHRECSIM}}(stk_s)$

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 $\text{VERIFY}(X, \sigma)$ then return false
    $\mathsf{bad} \leftarrow \mathsf{true}$
    Return false
If $(i_r = j')$ then
  If not $\text{VERIFY}(X, \sigma)$ 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

RATSENDSIM

If $\mathsf{op}[i_s] = \perp$ then $\mathsf{op}[i_s] \leftarrow$ “ch”
$x \leftarrow\$ \mathbb{Z}_p$
If $i_s = j'$ then $\sigma \leftarrow \text{TAG}(g^x)$
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

EXPSIM

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

CHSENDSIM

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 $\mathsf{rkey}[i_s]$

CHRECSIM

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 $\mathsf{rkey}[i_r]$

Fig. 11. Adversary $\mathcal{F}$ for proof of Theorem 1.

and $\Pr[\text{G}_2]$ by the advantage of the Diffie-Hellman adversary $\mathcal{O}_2$ that is defined in Fig. 12. Specifically, we have $\mathsf{Adv}^{\mathsf{odhe}}_{\mathbb{G},\mathsf{H},\mathcal{O}_2} = \Pr[\text{G}_1] - \Pr[\text{G}_2]$. 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. 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. □

Adversary $\mathcal{O}_2^{\text{UP,CH,EXP,HASH}}(hk, g, Y)$

$b \leftarrow\$ \{0,1\}$; $i_s \leftarrow 0$; $i_r \leftarrow 0$; $\mathsf{unchanged} \leftarrow \mathsf{true}$
$k_s[0] \leftarrow\$ \{0,1\}^{\mathsf{RKE.kl}}$; $k_r \leftarrow k_s[0]$
$fk_s[0] \leftarrow\$ \{0,1\}^{\mathsf{F.kl}}$; $fk_r \leftarrow fk_s[0]$; $hk \leftarrow\$ \{0,1\}^{\mathsf{H.kl}}$
$g \leftarrow\$ \mathbb{G}^*$; $y \leftarrow\$ \mathbb{Z}_p$; $stk_s \leftarrow (hk, g, Y)$
$b' \leftarrow\$ \mathcal{D}^{\text{RATSENDSIM,RATRECSIM,EXPSIM,CHSENDSIM,CHRECSIM}}(stk_s)$
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 $\mathsf{rkey}[i_r]$

Fig. 12. Adversary $\mathcal{O}_2$ for proof of Theorem 1.

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 $\mathsf{RE.IKg}$, $\mathsf{RE.SKg}$, $\mathsf{RE.RKg}$, $\mathsf{RE.Enc}$ and $\mathsf{RE.Dec}$, where $\mathsf{RE.Enc}$ and $\mathsf{RE.Dec}$ are deterministic. Associated to RE is a nonce space $\mathsf{RE.NS}$, sender's key generation randomness space $\mathsf{RE.RS}$, and a ciphertext length function $\mathsf{RE.cl}\colon \mathbb{N} \to \mathbb{N}$. Initial key generation algorithm $\mathsf{RE.IKg}$ 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 $\mathsf{RE.SKg}$ 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 $\mathsf{RE.RKg}$ 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 $\mathsf{RE.Enc}$ 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 $\mathsf{RE.Dec}$ 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 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 $\mathsf{Adv}^{\mathsf{rae}}_{\mathsf{RE},\mathcal{A}} = 2\Pr[\text{RAE}^{\mathcal{A}}_{\mathsf{RE}}] - 1$.

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 $\mathsf{RE} = \mathsf{RATCHET\text{-}ENC}[\mathsf{RKE}, \mathsf{SE}]$ with $\mathsf{RE.NS} = \mathsf{SE.NS}$, $\mathsf{RE.RS} = \mathsf{RKE.RS}$ and $\mathsf{RE.cl} = \mathsf{SE.cl}$ as follows. Let $\mathsf{RE.IKg} = \mathsf{RKE.IKg}$, $\mathsf{RE.SKg} = \mathsf{RKE.SKg}$, $\mathsf{RE.RKg} = \mathsf{RKE.RKg}$, $\mathsf{RE.Enc} = \mathsf{SE.Enc}$, and $\mathsf{RE.Dec} = \mathsf{SE.Dec}$. Thus RE is directly using RKE for key generation and SE for encryption.

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. The proof is in [7].

Theorem 2. *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* $q_{\textsc{RatSend}}$ *queries to its* RatSend *oracle,* $q_{\textsc{RatRec}}$ *queries to its* RatRec *oracle,* $q_{\textsc{Exp}}$ *queries to its* Exp *oracle,* $q_{\textsc{Enc}}$ *queries to its* Enc *oracle, and* $q_{\textsc{Dec}}$ *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 $\mathcal{A}$.*

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* [20, 16] used in multiple real-world messaging applications, such as in WhatsApp [26] and in the Secret Conversations mode of Facebook Messenger [17]. 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. Bellare, Jaeger and Stepanovs were supported in part by NSF grants CNS-1526801 and CNS-1228890, ERC Project ERCC FP7/615074 and a gift from Microsoft.

References

1. M. Abdalla, M. Bellare, and P. Rogaway. The oracle Diffie-Hellman assumptions and an analysis of DHIES. *CT-RSA 2001*.
2. M. Bellare, S. Duan, and A. Palacio. Key insulation and intrusion resilience over a public channel. *CT-RSA 2009*.
3. M. Bellare, D. Pointcheval, and P. Rogaway. Authenticated key exchange secure against dictionary attacks. *EUROCRYPT 2000*.
4. M. Bellare and P. Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. *ACM CCS 1993*.
5. M. Bellare and P. Rogaway. Entity authentication and key distribution. *CRYPTO 1993*.

6. M. Bellare and P. Rogaway. The security of triple encryption and a framework for code-based game-playing proofs. *EUROCRYPT 2006*.
7. M. Bellare, A. C. Singh, J. Jaeger, M. Nyayapati, and I. Stepanovs. Ratcheted encryption and key exchange: The security of messaging. Cryptology ePrint Archive, Report 2016/1028, 2016.
8. M. Bellare and B. Tackmann. The multi-user security of authenticated encryption: AES-GCM in TLS 1.3. *CRYPTO 2016, Part I*.
9. M. Bellare and B. S. Yee. Forward-security in private-key cryptography. *CT-RSA 2003*.
10. N. Borisov, I. Goldberg, and E. Brewer. Off-the-record communication, or, why not to use pgp. *Proceedings of the ACM Workshop on Privacy in the Electronic Society (WPES), 2004*.
11. K. Cohn-Gordon, C. Cremers, B. Dowling, L. Garratt, and D. Stebila. A formal security analysis of the Signal messaging protocol. *Proc. IEEE European Symposium on Security and Privacy (EuroS&P) 2017*.
12. K. Cohn-Gordon, C. Cremers, and L. Garratt. On post-compromise security. *IEEE 29th Computer Security Foundations Symposium (CSF), 2016*.
13. Y. Dodis, J. Katz, S. Xu, and M. Yung. Key-insulated public key cryptosystems. *EUROCRYPT 2002*.
14. Y. Dodis, J. Katz, S. Xu, and M. Yung. Strong key-insulated signature schemes. *PKC 2003*.
15. Y. Dodis, W. Luo, S. Xu, and M. Yung. Key-insulated symmetric key cryptography and mitigating attacks against cryptographic cloud software. *ASIACCS 2012*.
16. T. P. (editor) and M. Marlinspike. The double ratchet algorithm. `https://whispersystems.org/docs/specifications/doubleratchet/`, Nov. 20, 2016.
17. Facebook. Messenger secret conversations technical whitepaper. Technical whitepaper, `https://fbnewsroomus.files.wordpress.com/2016/07/secret_conversations_whitepaper-1.pdf`, July 8, 2016.
18. M. Fischlin and F. Günther. Multi-stage key exchange and the case of Google's QUIC protocol. *ACM CCS 2014*.
19. A. Langley. Pond. GitHub repository, README.md, `https://github.com/agl/pond/commit/7bb06244b9aa121d367a6d556867992d1481f0c8`, 2012.
20. M. Marlinspike. Advanced cryptographic ratcheting. `https://whispersystems.org/blog/advanced-ratcheting/`, Nov. 26, 2013.
21. T. Okamoto and D. Pointcheval. The gap-problems: A new class of problems for the security of cryptographic schemes. *PKC 2001*.
22. Open Whisper Systems. Signal protocol library for java/android. GitHub repository, `https://github.com/WhisperSystems/libsignal-protocol-java`, 2017.
23. P. Rogaway. Authenticated-encryption with associated-data. *ACM CCS 2002*.
24. P. Rogaway. Nonce-based symmetric encryption. *FSE 2004*.
25. N. Unger, S. Dechand, J. Bonneau, S. Fahl, H. Perl, I. Goldberg, and M. Smith. SoK: Secure messaging. *IEEE Symposium on Security and Privacy, 2015*.
26. WhatsApp. Whatsapp encryption overview. Technical white paper, `https://www.whatsapp.com/security/WhatsApp-Security-Whitepaper.pdf`, Apr. 4, 2016.