# Ratcheted Encryption and Key Exchange: The Security of Messaging

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

February 2016

**Abstract**

We give a theoretical treatment of ratcheting, lifting it from a technique used in secure messaging protocols to a cryptographic primitive. To allow a modular treatment, we decouple the creation of keys from their use. We define ratcheted key exchange and give a protocol proven to achieve it. We then define ratcheted encryption and show how to achieve it generically from ratcheted key exchange and standard encryption.

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

[2] Salesforce. Work done while at UCSD. Email: acampers@eng.ucsd.edu.

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

[4] Salesforce. Work done while at UCSD. Email: mnyayapa@ucsd.edu.

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

# Contents

# 1 Introduction

In 2015, the number of text messages sent worldwide reached 8.3 trillion (Portio Research). This year, WhatsApp alone reports handling 42 billion text messages *per day*. Protecting the privacy and integrity of all this data is urgent. Providers recognize this, and a large number of secure-messaging apps have emerged [1], for example: WhatsApp, Viber, LINE, Telegram, Kakao Talk, Signal Private Messenger, Cyber Dust, ChatSecure, Threema, Wickr Secure Messenger, Cyphr, CoverMe, iMessage and GoldBug.

With these offerings, providers are seizing the opportunity to provide security that aims to be *better*, in several ways, than the security than is currently provided by TLS-based web services. The first way it aims to be better is in being end-to-end. In Gmail, the connection between users and Google is secured (via TLS), but Google has the data in the clear. Secure messaging apps in contrast encrypt messages under the keys of the communicants and even the server does not see data in the clear. The second way it aims to be better is through the use of a technique called *ratcheting* that changes keys with high frequency. Ratcheting, in some form, is used in almost all the above-mentioned secure-messaging apps, and has an importance peculiar to this domain due to the long-lived nature of messaging conversations.

This paper initiates a theoretical treatment of ratcheting. We aim to figure out and formalize the goals and give proven-secure schemes. The topic, we found, is more complex than it may seem.

Ratcheting. Secure messaging (as with TLS), uses the ubiquitous paradigm in which (1) the parties run an (explicit or implicit) authenticated key exchange to get a session key $K$, and then (2) secure (encrypt and authenticate) data using the session key $K$. Ratcheting is employed in (2). We will not be concerned with (1). Our setting is thus a simple symmetric one of two parties starting with a shared key $K$, and we are concerned with secure communication under it.

In TLS, all data is secured under $K$ with an authenticated encryption scheme. Under ratcheting, the key is constantly changing. The method was introduced by Borisov, Goldberg and Brewer (BGB) [12] in their highly influential OTR (Off the Record) Communication system. (They do not call it ratcheting, this term originating later.) Roughly it works like this:

$$B \to A\text{: } g^{b_1} \ ; \ A \to B\text{: } g^{a_1}, \mathcal{E}(K_1, M_1) \ ; \ B \to A\text{: } g^{b_2}, \mathcal{E}(K_2, M_2) \ ; \ \ldots \tag{1}$$

Here $a_i$ and $b_i$ are random exponents picked by $A$ and $B$ respectively; $K_1 = H(g^{b_1 a_1})$, $K_2 = H(g^{a_1 b_2})$, ...; $H$ 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.

Different messaging apps specify and implement different variants and extensions. Exactly what security goals these types of methods target or achieve is nebulous. In their SOK (Systemization of Knowledge) paper on secure messaging, UDBFPGS [21] survey many of the systems that existed at the time and attempt to classify them in terms of security. They note that security claims about ratcheting in different places include "forward-secrecy," "backward-secrecy," "self-healing" and "future secrecy." They conclude: *"The terms are controversial and vague in the literature"* [21, Section 2.D].

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. This was already a consider-

able challenge. 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). Formalizing this involves figuring out what is the best achievable and gets quite delicate. Amongst the issues is that following exposure there is some (necessary) time lag before security is regained, and that privacy and integrity are related. For ratcheted key exchange, un-exposed keys are required to be indistinguishable from random in the spirit of [8] —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 [19], so that authenticity in particular is provided. The definitions are strong, asking for the most that might be achievable.

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 [10]. 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 [12] and the in-use protocols, but there are small but important differences, including MAC-based authentication of the transferred ephemeral $g^a$ values and the way keys are derived. Proving that the protocol meets our definition of secure ratcheted key-exchange turns out to be challenging. Our proof is based on the SCDH (Strong Computational Diffie-Hellman) assumption [4] and is in the random-oracle model [7]. It is obtained in two steps. The first, that we give, is a standard-model reduction to an assumption we call ODHE (Oracle Diffie-Hellman with Exposures). The second, that can be obtained via techniques from [7] and we accordingly omit, is a validation of ODHE under SCDH in the ROM.

We treat the core ratcheting problem, that we call one-sided. (One party is a sender and the other a receiver, rather than both playing both roles.) This already involves considerable complexity. We will briefly discuss extensions to two-sided ratcheting as well as to double-ratcheting [3].

Model and syntax. Our syntax specifies a scheme RKE for ratcheted key exchange via three algorithms: initial key generation RKE.IKg, sender key generation RKE.SKg and receiver key generation RKE.RKg, modeling the following process. (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 RKE.SKg on its current keys to get an ephemeral public key that it sends to the receiver, as well as updated keys for itself. The receiver $B$ correspondingly will run RKE.RKg on a received ephemeral public key and its current keys to get updated keys, transmitting nothing. RKE.IKg provides initial output keys (the same for both parties) and initial session keys. These in practice would be derived from the assumed shared (authentic) symmetric key given by a higher-level process that RKE.IKg allows us to remove from the picture so as to isolate ratcheting as a stand-alone primitive. We

additionally have RKE.IKg issue the receiver a secret key *lsk* whose associated public key *lpk* is held by the sender. This is not a "real" public key, by which we mean that it does not need a certificate. It is rather a convenient abstraction boundary, representing the first DH value $lpk = g^b$ sent by the receiver in Equation (1). In practice it should be sent authenticated. A ratcheted encryption scheme RE maintains the same three key-generation algorithms, now denoted RE.IKg, RE.SKg and RE.RKg, and adds an encryption algorithm RE.Enc for the sender —in the nonce-based vein [19], taking a key, nonce, message and header to deterministically return a ciphertext— and a corresponding decryption algorithm RE.Dec for the receiver. The key for encryption and decryption is what ratcheted key exchange referred to as the output key.

The correctness requirement is more demanding than is usual, asking for some robustness: if the receiver receives an incorrect ephemeral public key that it rejects, it maintains its state and will still accept a subsequent correct ephemeral public key. This prevents a denial-of-service attack in which a single incorrect ephemeral public key sent to the receiver results in all future communications being rejected.

Security. In the spirit of [8] we give the adversary complete control of communication. Our definition of security for ratcheted key exchange in Section 4.2 is via a single game KIND that captures both privacy and integrity. 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 ephemeral public key) can result in continued violation of integrity. Our game carefully makes the necessary restrictions to capture these and other situations, ultimately asking for the best possible security. For ratcheted encryption, we again give in Section 5 a single game RAE capturing 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 [12] or WhatsApp, but analysis is quite involved. The receiver long term public key is a group element $g^b$. An ephemeral public key is a $g^a$ *accompanied by a mac under the prior session key.* We explain why the mac is crucial to security. The output and next session key are derived via a hash function applied to $g^{ab}$. Theorem 4.1 establishes that the scheme meets our stringent notion of security for ratcheted key exchange. The proof uses a game sequence that includes a hybrid argument to reduce the security of the ratcheted key exchange to our ODHE (Oracle Diffie-Hellman with Exposures) assumption. The latter is an extension of the ODH assumption of [4] and, like the latter, can be validated in the ROM under the SCDH assumption of [4], which in turn is a variant of the Gap-DH assumption of [18]. Ultimately, this yields a proof of security for our ratcheted key exchange protocol under the SCDH assumption in the ROM.

Our construction of a ratcheted encryption scheme in Section 5 is a generic combination of any ratcheted key exchange scheme (meeting our definition of security) and any nonce-based authenticated encryption scheme. Theorem 5.1 establishes that the scheme meets our notion of security for ratcheted encryption. The analysis is facilitated by assuming multi-user security for the base nonce-based encryption scheme as defined in [10], but a hybrid argument reduces this to the standard

single-user security defined in [19]. Encryption schemes meeting this notion are readily available.

Discussion and related work. 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. This is to some extent outside our scope, since messaging-app providers taking the threat seriously enough to implement ratcheting motivates cryptographers seriously evaluating ratcheting, but is still worth discussion. 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.

Whisper Systems's Signal protocol [3] uses double-ratcheting— if $A$ sends multiple messages before receiving a reply, it will send a single $g^a$, and then advance its key for every message by hashing the prior key. Most current secure messaging apps adapt the Signal protocol. Double ratcheting is effectively a combination of two studied primitives, the first being our (single) ratcheting, and the other being forward-secure pseudorandom generators and key exchange [11], and might be treated in this way. We do not undertake such a treatment here. Messaging apps also implement two-sided ratcheting, where the parties alternate sender and receiver roles and the DH values are intertwined as in Equation (1). We focused on one-sided ratcheting as more basic and already complex enough. Treating the two-sided version would require an extension of our work.

Our work was instigated by WhatsApp's white paper [22] and their announcement of their move to default end-to-end security. If ratcheting was to be used 42 billion times a day, we'd like to know precisely what it does. In the end, our work represents our view of what ratcheting aims to do and how to do it securely, but we stop well short of claiming an analysis or proof of what WhatsApp actually implements. Their full system is not only more complex than ours but we are not even sure exactly what it is. We did not feel we got a fully detailed specification from the white paper [22]. We were also confused by the latter contradicting pseudocode [2]. In our work we have thus stepped back and, in the tradition of cryptographic research, given ourselves the freedom to deviate somewhat from what may be implemented, or even be in designer's minds, to study, understand, formalize and achieve ratcheting as a goal in its own right.

Key-insulated cryptography [14, 15, 16] 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 [6]. Ratcheting in contrast works in a model where all communication is under adversary control.

Cohn-Gordon, Cremers and Garratt [13] 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.

## 2 Preliminaries

Notation and conventions. Let $\mathbb{N} = \{0, 1, 2, \ldots\}$ be the set of non-negative integers. Let $\varepsilon$ 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 $\mathsf{mem}$ 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

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

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

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

Function families. A family of functions $\mathsf{F}$ specifies a deterministic algorithm $\mathsf{F.Ev}$. Associated to $\mathsf{F}$ is a key length $\mathsf{F.kl} \in \mathbb{N}$, an input set $\mathsf{F.In}$, and an output set $\mathsf{F.Out}$. 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 $y \in \mathsf{F.Out}$.

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

Symmetric encryption schemes. A symmetric encryption scheme $\mathsf{SE}$ specifies deterministic algo-

rithms $\mathsf{SE.Enc}$ and $\mathsf{SE.Dec}$. Associated to $\mathsf{SE}$ is a key length $\mathsf{SE.kl} \in \mathbb{N}$, a nonce space $\mathsf{SE.NS}$, and a ciphertext length function $\mathsf{SE.cl}\colon \mathbb{N} \to \mathbb{N}$. Encryption algorithm $\mathsf{SE.Enc}$ takes $sk \in \{0,1\}^{\mathsf{SE.kl}}$, a nonce $n \in \mathsf{SE.NS}$, a message $m \in \{0,1\}^*$ and a header $h \in \{0,1\}^*$ to return a ciphertext $c \in \{0,1\}^{\mathsf{SE.cl}(|m|)}$. Decryption algorithm $\mathsf{SE.Dec}$ takes $sk, n, c, h$ to return message $m \in \{0,1\}^* \cup \{\perp\}$, where $\perp$ denotes incorrect decryption. Decryption correctness requires that $\mathsf{SE.Dec}(sk, n, \mathsf{SE.Enc}(sk, n, m, h), h) = m$ for all $sk \in \{0,1\}^{\mathsf{SE.kl}}$, all $n \in \mathsf{SE.NS}$, all $m \in \{0,1\}^*$, and all $h \in \{0,1\}^*$. In this work we consider only *nonce-based* encryption schemes *with associated data*, and we therefore omit stating these qualifiers. Nonce-based symmetric encryption was introduced in [20], whereas [19] also considers it in the setting with associated data.

Multi-user authenticated encryption. Consider game MAE of Fig. 1, associated to a symmetric encryption scheme $\mathsf{SE}$ and an adversary $\mathcal{N}$. It extends the definition of authenticated encryption with associated data for nonce-based schemes [19] to the multi-user setting, the latter case first formalized in [10]. The adversary is allowed to increase the number of users by calling oracle New. For any of the generated user keys, $\mathcal{N}$ can request encryptions of plaintext messages by calling oracle Enc and decryptions of ciphertexts by calling oracle Dec. In the real world Enc and Dec provide correct encryptions and decryptions, whereas in the random world Enc returns uniformly random ciphertexts and Dec always 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. We allow $\mathcal{N}$ to call Enc only once for every unique user-nonce pair $(i, n)$. The definition could be extended to not allow repeated queries of $(i, n, m, h)$, but this requirement is satisfied by fewer schemes. The advantage of $\mathcal{N}$ in breaking the MAE security of $\mathsf{SE}$ 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 assumption [5] in a cyclic group requires that it is hard to distinguish between a random string and a hash function $H$ applied to $g^{xy}$ 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 to allow multiple queries (with exposures) and broader classes of inputs to the hash function.

Oracle Diffie-Hellman assumption with exposures. Let $\mathbb{G}$ be a cyclic group of order $p \in \mathbb{N}$, and let $\mathbb{G}^*$ denote the set of its generators. Let $\mathsf{H}$ 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 $x$ and returns $g^x$, along with increasing the integer counter $i_0$ that denotes the number of the current challenge exponent (indexed from 0). 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 of the challenge exponents $x$, adversary can choose to either call oracle Exp to get the value of $x$, or call oracle Ch with input $s$ to get a challenge value that is generated as follows. In the real world (when $b = 1$) oracle Ch returns $\mathsf{H.Ev}(hk, i_0 \,\|\, s \,\|\, g^{xy})$, and in the random world (when $b = 0$) it returns a uniformly random string. The goal of the adversary is to distinguish between these two cases. In order to avoid trivial attacks, $\mathcal{O}$ is not allowed to make equivalent queries to oracles Ch and Hash. Note that adversary is allowed to win the game if it happens to guess a future challenge exponent $x$ and query it to

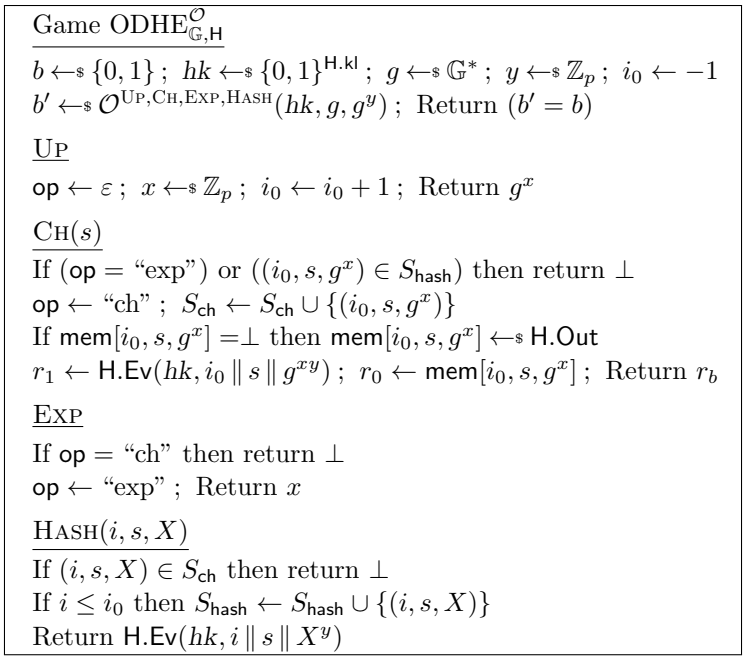


Figure 2: Game defining oracle Diffie-Hellman assumption with exposures in $\mathbb{G}, \mathsf{H}$.

oracle Hash ahead of time (the corresponding triple $(i, s, X)$ will not be added to the set of inputs that are not allowed to be made to oracle Ch). Finally, our definition of string concatenation $\|$ requires that the result of $i_0 \,\|\, s \,\|\, g^{xy}$ is a uniquely decodable string, meaning that the encoding of $(i, s, g^{xy})$ into strings is injective. The advantage of $\mathcal{O}$ in breaking the ODHE security of $\mathbb{G}, \mathsf{H}$ is defined as $\mathsf{Adv}^{\mathsf{odhe}}_{\mathbb{G},\mathsf{H},\mathcal{O}} = 2\Pr[\text{ODHE}^{\mathcal{O}}_{\mathbb{G},\mathsf{H}}] - 1$.

Plausibility of the assumption. We do not know of any standard model group and function family which can be shown to achieve this security notion. The original oracle Diffie-Hellman assumption of [5] was justified by a reduction in the random oracle model to the strong Diffie-Hellman assumption. The latter was defined in [5] and is a weaker version of the gap Diffie-Hellman assumption from [18]. We note that an analogous result could be obtained for our expanded oracle Diffie-Hellman notion. Therefore, we do not provide the proof of this in our work.

# 4 Ratcheted key exchange

Ratcheted key exchange allows users to agree on shared secret keys while providing very strong security guarantees. In this work we consider a setting that encompasses two parties, and we assume that only one of them sends key agreement messages. We call this party a sender, and the other party – a receiver. We also assume that the receiver owns a key pair consisting of a public key and a secret key, such that the public key is available to the sender. This enables us to make the first steps towards modeling 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 consider the group chat setting where multiple users engage in shared conversations.

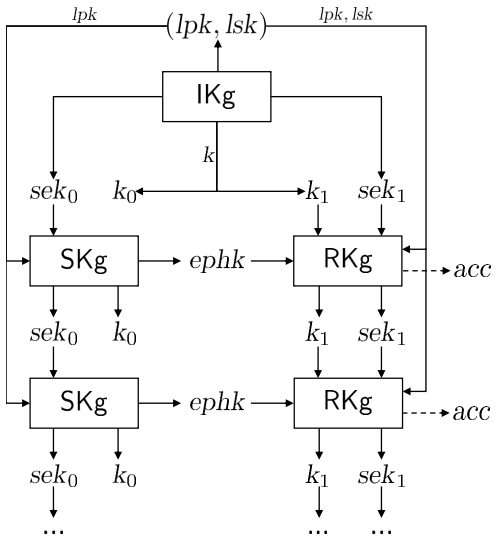


Figure 3: **The interaction between ratcheted key exchange algorithms.**

---

### 4.1 Definition of ratcheted key exchange

Consider Fig. 3 for an informal overview of algorithms that constitute a racheted key exchange scheme RKE, and the interaction between them. We define three algorithms RKE.IKg, RKE.SKg and RKE.RKg as follows. Initial key generation algorithm RKE.IKg generates and distributes the following keys. Receiver's key pair $(lpk, lsk)$ consists of a long-term public key $lpk$ and a long-term secret key $lsk$. 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. Session keys $sek_0$ and $sek_1$ 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 $lpk, sek_0, k_0$ and the receiver gets $lsk, sek_1, k_1$, where $k_0 = k_1 = k$.

Whenever the sender wants to generate a new shared secret key, it runs sender's key generation algorithm RKE.SKg. Algorithm RKE.SKg takes receiver's long-term public key $lpk$ and sender's session key $sek_0$. It generates a new output key $k_0$, updates sender's session key $sek_0$, and outputs an ephemeral key $ephk$. The latter can be used by the receiver in order to generate the same output key.

Receiver's key generation algorithm RKE.RKg takes long-term public key $lpk$, long-term secret key $lsk$, receiver's session key $sek_1$, ephemeral key $ephk$ (received from the sender) and the current shared output key $k_1$. It returns a (possibly new) output key $k_1$, updates receiver's session key $sek_1$, and sets a Boolean flag $acc$ indicating whether a new output key was succesfully generated. The correctness of key generation is two-fold. First, if the receiver gets a valid ephemeral key $ephk$, then algorithm RKE.RKg must set $acc = \mathsf{true}$ and produce a new receiver's output key $k_1$ that is equal to the corresponding sender's output key $k_0$. Otherwise, RKE.RKg must set $acc = \mathsf{false}$ and leave the keys $k_1, sek_1$ unchanged; this requirement is the reason why RKE.RKg takes the old value of $k_1$ as one of its inputs. We now formalize the described scheme and its correctness requirements.

Ratcheted key exchange schemes. A ratcheted key exchange scheme RKE specifies algorithms RKE.IKg, RKE.SKg and RKE.RKg. Associated to RKE is an output key length RKE.kl $\in \mathbb{N}$ and sender's key

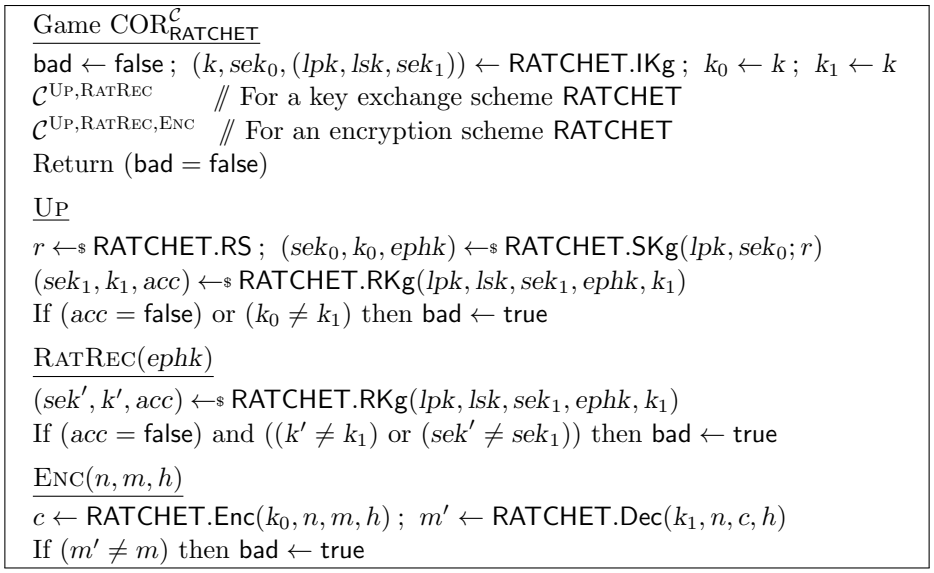


Figure 4: Game simultaneously defining correctness of ratcheted key exchange scheme RATCHET, and correctness of ratcheted encryption scheme RATCHET. Encryption oracle Enc is defined only for the latter case.

---

generation randomness space RKE.RS. Initial key generation algorithm RKE.IKg returns $k, sek_0$, $(lpk, lsk, sek_1)$, where $k \in \{0,1\}^{\mathsf{RKE.kl}}$ is an output key, $sek_0$ is a sender's session key, and $lpk, lsk, sek_1$ are receiver's long-term public key, receiver's long-term secret key and receiver's session key, respectively. Sender's key generation algorithm RKE.SKg takes $lpk, sek_0$ and randomness $r \in \mathsf{RKE.RS}$ to return a new sender's session key $sek_0$, a new sender's output key $k_0 \in \{0,1\}^{\mathsf{RKE.kl}}$, and an ephemeral key $ephk$. Receiver's key generation algorithm RKE.RKg takes $lpk, lsk, sek_1, ephk$ and receiver's output key $k_1 \in \{0,1\}^{\mathsf{RKE.kl}}$ to return a new receiver's session key $sek_1$, a new receiver's output key $k_1 \in \{0,1\}^{\mathsf{RKE.kl}}$, and a flag $acc \in \{\mathsf{true}, \mathsf{false}\}$.

Correctness of ratcheted key exchange. Consider game COR of Fig. 4 associated to a ratcheted key exchange scheme RATCHET and an adversary $\mathcal{C}$, where $\mathcal{C}$ is provided with an access to oracles Up and RatRec. Oracle Up runs algorithm RKE.SKg to generate a new sender's output key $k_0$ with the corresponding ephemeral key $ephk$; it then runs RKE.RKg with $ephk$ as input to generate a new receiver's output key $k_1$. It is required that $acc = \mathsf{true}$ and $k_0 = k_1$ at the end of every Up call. Oracle RatRec takes an ephemeral key $ephk$ of adversary's choice and attempts to run RKE.RKg with $ephk$ (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 $k_1, sek_1$ remain unchanged. We consider an unbounded adversary and allow it to call its oracles in any order. The advantage of $\mathcal{C}$ breaking the correctness of RATCHET is defined as $\mathsf{Adv}^{\mathsf{cor}}_{\mathsf{RATCHET},\mathcal{C}} = 1 - \Pr[\mathrm{COR}^{\mathcal{C}}_{\mathsf{RATCHET}}]$. Correctness property requires that $\mathsf{Adv}^{\mathsf{cor}}_{\mathsf{RATCHET},\mathcal{C}} = 0$ for all unbounded adversaries $\mathcal{C}$.

### 4.2 Security of ratcheted key exchange

Ratcheted key exchange attempts to provide strong security guarantees even in the presence of an adversary that can steal the secrets stored by the sender. We consider an active attacker who is able to intercept and modify any ephemeral keys sent between the sender and the receiver. The

goal is that the attacker cannot make them agree on different secret keys or distinguish these secret keys from random strings. Furthermore, we desire certain stronger security properties to hold even if an attacker manages to steal the secrets stored by sender, which we refer to as forward security and future security. Forward security requires that such an attacker cannot distinguish prior keys from random. Future security requires that the knowledge of sender secrets at the current time period can not be used to distinguish future validly generated keys from random strings.

It is clear that if an attacker steals the secret information of the sender it can create its own ephemeral keys resulting in the receiver agreeing on a "secret" key which is known by the attacker. At this point, it can be difficult to say what restrictions should be placed on the keys the attacker makes the receiver agree to. For example, is it a further breach of security if the attacker then later causes the sender and receiver to agree on the same secret key? What should happen if the attacker later sends a validly generated ephmeral key to the receiver?

In our security model we choose to insist on two straightforward policies in this scenario. The first is that whenever a non-validly generated ephemeral key is accepted by the receiver (which we emphasize should only be possible when an attacker has stolen the sender's secrets) then even full knowledge of the key the receiver has generated should not leak any information about other correctly generated keys. The second is that at any point in time, if a validly generated ephemeral key is accepted by the receiver then it should agree with the sender on what the key is and the adversary should not be able to distinguish it 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 $\mathsf{RKE}$ and an adversary $\mathcal{D}$. The advantage of $\mathcal{D}$ in breaking the KIND security of $\mathsf{RKE}$ is defined as $\mathsf{Adv}^{\mathsf{kind}}_{\mathsf{RKE},\mathcal{D}} = 2\Pr[\mathrm{KIND}^{\mathcal{D}}_{\mathsf{RKE}}] - 1$.

The adversary is given receiver's long-term public key $lpk$ as well as access to oracles RatSend, RatRec, Exp, ChSend, and ChRec. It can call oracle RatSend to receive validly generated ephemeral keys $ephk$ from the sender, and it can call oracle RatRec to pass arbitary ephemeral keys to the receiver. Oracle Exp returns the current secrets $sek_0, k_0$ possessed by the sender as well as the random seed $r$ that was used to create the most recent $ephk$ in RatSend.

The challenge oracles ChSend and ChRec provide the adversary with keys $k_0$ and $k_1$ 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 $\mathsf{op}$ and $\mathsf{auth}$ (initialized as empty) as well as a boolean flag $\mathsf{restricted}$ (initialized as false). Specifically, $\mathsf{op}$ 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 $\mathsf{auth}$ keeps track of the ephemeral keys $ephk$ validly generated by RatSend so that we can set the flag $\mathsf{restricted}$ whenever the adversary has taken advantage of an Exp query to send maliciouly generated $ephk$ to RatRec. In this case we do not expect the receiver's key $k_1$ to look random or match the sender's key $k_0$ so ChRec is "restricted" and will return $k_1$ 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 a malicious ephemeral key $ephk^*$ that is accepted by the receiver, while not setting the $\mathsf{restricted}$ flag to $\mathsf{true}$. By making the receiver accept $ephk^*$, the adversary achieves the situation when the sender and the receiver produce different output keys $k_0 \neq k_1$. Now adversary can call oracles ChSend and ChRec to get both keys and compare them to win the game. In the real

world (when $b = 1$) the returned keys will be different, whereas in the random world (when $b = 0$) they will be the same.

Allowing recovery from failures. Consider a situation when an attacker steals all sender's secrets. At this point, the attacker has an ability to impersonate the sender. It would 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 ephemeral keys *ephk*. This sets the restricted flag to true, making the CHREC oracle always return the real receiver's key $k_1$ 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 the adversary keeps sending maliciously generated ephemeral keys *ephk* we keep restricted set to true, thereby requiring that real receiver's key $k_1$ returned from CHREC is of no help in distinguishing the real sender's key $k_0$ from random as desired from the first policy. To match the second policy, if the adversary the validly generated *ephk* (i.e. $ephk = \mathsf{auth}[i_1]$) to RATREC and it is accepted, the restricted flag is set back to false so that the output of CHREC again depends on the bit, thus requiring $k_1$ to be equal to $k_0$ and indistinguishable from random.

### 4.3 Construction of ratcheted key exchange

In this section we construct a ratcheted key exchange scheme, and deduce a bound on the success of any adversary attacking its KIND security. The idea of our construction is as follows. We let the sender and the receiver perform the Diffie-Hellman key exchange. The receiver's long-term secret key contains a secret DH exponent $lsk = y$, and its long-term public key contains the corresponding public value $lpk = g^y$ (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 $k_0 = H(lpk^x) = H(g^{xy})$, where $H$ is some hash function. The sender then sends an ephemeral key containing $g^x$ to the receiver, enabling the latter to compute the same output key. In order to ensure the authenticity of key exchange, both parties use a shared MAC key, meaning the ephemeral key 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 future 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 $\mathsf{F}$ be a function family such that $\mathsf{F.In} = \mathbb{G}$. Let $\mathsf{H}$ be a function family such that $\mathsf{H.In} = \{0,1\}^*$ and $\mathsf{H.Out} = \{0,1\}^\kappa$ for some $\kappa > \mathsf{F.kl}$. 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 $\mathsf{RKE.kl} = \kappa - \mathsf{F.kl}$ and $\mathsf{RKE.RS} = \mathbb{Z}_p$.

Design considerations. 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 ephemeral key $ephk_i = (g^{x_i}, \sigma_i)$. Including $i$ in $w$ ensures that an attaker cannot cause the sender and recevier to agree on the same keys in different

Game $\text{KIND}^{\mathcal{D}}_{\mathsf{RKE}}$

$b \leftarrow\!\!\$\ \{0,1\}$ ; $i_0 \leftarrow 0$ ; $i_1 \leftarrow 0$
$(k, sek_0, (lpk, lsk, sek_1)) \leftarrow\!\!\$\ \mathsf{RKE.IKg}$
$k_0 \leftarrow k$ ; $k_1 \leftarrow k$
$b' \leftarrow\!\!\$\ \mathcal{D}^{\text{RatSend},\text{RatRec},\text{Exp},\text{ChSend},\text{ChRec}}(lpk)$
Return $(b' = b)$

RatSend
$r \leftarrow\!\!\$\ \mathsf{RKE.RS}$
$(sek_0, k_0, ephk) \leftarrow \mathsf{RKE.SKg}(lpk, sek_0; r)$
$\mathsf{auth}[i_0] \leftarrow ephk$ ; $i_0 \leftarrow i_0 + 1$
Return $ephk$

RatRec($ephk$)
$z \leftarrow\!\!\$\ \mathsf{RKE.RKg}(lpk, lsk, sek_1, ephk, k_1)$
$(sek_1, k_1, acc) \leftarrow z$
If not $acc$ then return $\mathsf{false}$
If $\mathsf{op}[i_1] =$ "exp" then $\mathsf{restricted} \leftarrow \mathsf{true}$
If $ephk = \mathsf{auth}[i_1]$ then $\mathsf{restricted} \leftarrow \mathsf{false}$
$i_1 \leftarrow i_1 + 1$ ; Return $\mathsf{true}$

Exp
If $\mathsf{op}[i_0] =$ "ch" then return $\perp$
$\mathsf{op}[i_0] \leftarrow$ "exp" ; Return $(r, sek_0, k_0)$

ChSend
If $\mathsf{op}[i_0] =$ "exp" then return $\perp$
$\mathsf{op}[i_0] \leftarrow$ "ch"
If $\mathsf{rkey}[i_0] = \perp$ then $\mathsf{rkey}[i_0] \leftarrow\!\!\$\ \{0,1\}^{\mathsf{RKE.kl}}$
If $b = 1$ then return $k_0$ else return $\mathsf{rkey}[i_0]$

ChRec
If $\mathsf{restricted}$ then return $k_1$
If $\mathsf{op}[i_1] =$ "exp" then return $\perp$
$\mathsf{op}[i_1] \leftarrow$ "ch"
If $\mathsf{rkey}[i_1] = \perp$ then $\mathsf{rkey}[i_1] \leftarrow\!\!\$\ \{0,1\}^{\mathsf{RKE.kl}}$
If $b = 1$ then return $k_1$ else return $\mathsf{rkey}[i_1]$

Game $\text{RAE}^{\mathcal{A}}_{\mathsf{RE}}$

$b \leftarrow\!\!\$\ \{0,1\}$ ; $i_0 \leftarrow 0$ ; $i_1 \leftarrow 0$
$(k, sek_0, (lpk, lsk, sek_1)) \leftarrow\!\!\$\ \mathsf{RE.IKg}$
$k_0 \leftarrow k$ ; $k_1 \leftarrow k$
$b' \leftarrow\!\!\$\ \mathcal{A}^{\text{RatSend},\text{RatRec},\text{Exp},\text{Enc},\text{Dec}}(lpk)$
Return $(b' = b)$

RatSend
$r \leftarrow\!\!\$\ \mathsf{RE.RS}$
$(sek_0, k_0, ephk) \leftarrow \mathsf{RE.SKg}(lpk, sek_0; r)$
$\mathsf{auth}[i_0] \leftarrow ephk$ ; $i_0 \leftarrow i_0 + 1$
Return $ephk$

RatRec($ephk$)
$z \leftarrow\!\!\$\ \mathsf{RE.RKg}(lpk, lsk, sek_1, ephk, k_1)$
$(sek_1, k_1, acc) \leftarrow z$
If not $acc$ then return $\mathsf{false}$
If $\mathsf{op}[i_1] =$ "exp" then $\mathsf{restricted} \leftarrow \mathsf{true}$
If $ephk = \mathsf{auth}[i_1]$ then $\mathsf{restricted} \leftarrow \mathsf{false}$
$i_1 \leftarrow i_1 + 1$ ; Return $\mathsf{true}$

Exp
If $\mathsf{op}[i_0] =$ "ch" then return $\perp$
$\mathsf{op}[i_0] \leftarrow$ "exp" ; Return $(r, sek_0, k_0)$

Enc($n, m, h$)
If $\mathsf{op}[i_0] =$ "exp" then return $\perp$
$\mathsf{op}[i_0] \leftarrow$ "ch"
If $(i_0, n) \in U$ then return $\perp$
$c_1 \leftarrow \mathsf{RE.Enc}(k_0, n, m, h)$
$c_0 \leftarrow\!\!\$\ \{0,1\}^{\mathsf{RE.cl}(|m|)}$ ; $U \leftarrow U \cup \{(i_0, n)\}$
$S \leftarrow S \cup \{(i_0, n, c_b, h)\}$
Return $c_b$

Dec($n, c, h$)
If $\mathsf{restricted}$ then
  Return $\mathsf{RE.Dec}(k_1, n, c, h)$
If $\mathsf{op}[i_1] =$ "exp" then return $\perp$
$\mathsf{op}[i_1] \leftarrow$ "ch"
If $(i_1, n, c, h) \in S$ then return $\perp$
$m \leftarrow \mathsf{RE.Dec}(k_1, n, c, h)$
If $b = 1$ then return $m$ else return $\perp$

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

orders. Below we describe an active "key-collision" attack against the KIND security of the scheme that is prevented by including $ephk_i$ in $w$. Finally, note that our concatenation operator $\|$ is defined to produce uniquely decodable strings, meaning that the encoding that maps $(i, \sigma_i, g^{x_i}, g^{x_i y})$ into a string $w$ is injective; this helps to avoid attacks that attempt to take advantage of malleable encodings.

Key-collision attacks. We now describe an attack idea that does not work against our construction but would have been possible if the ephmeral key $ephk$ were not included in the hash function.

Algorithm $\mathsf{RKE.IKg}$
$k \leftarrow\!\!\$\ \{0,1\}^{\mathsf{RKE.kl}}$
$fk \leftarrow\!\!\$\ \{0,1\}^{\mathsf{F.kl}}$
$hk \leftarrow\!\!\$\ \{0,1\}^{\mathsf{H.kl}}$
$g \leftarrow\!\!\$\ \mathbb{G}^*\ ;\ y \leftarrow\!\!\$\ \mathbb{Z}_p$
$lpk \leftarrow (hk, g, g^y)\ ;\ lsk \leftarrow y$
$sek_0 \leftarrow (0, fk)$
$sek_1 \leftarrow (0, fk)$
$z \leftarrow (k, sek_0, (lpk, lsk, sek_1))$
Return $z$

Algorithm $\mathsf{RKE.SKg}((hk, g, Y), (i_0, fk_0); r)$
$x \leftarrow r\ ;\ X \leftarrow g^x\ ;\ \sigma \leftarrow \mathsf{F.Ev}(fk_0, X)$
$s \leftarrow \mathsf{H.Ev}(hk, i_0 \,\|\, \sigma \,\|\, X \,\|\, Y^x)$
$k_0 \leftarrow s[1 \ldots \mathsf{RKE.kl}]$
$fk_0 \leftarrow s[\mathsf{RKE.kl}+1 \ldots \mathsf{RKE.kl}+\mathsf{F.kl}]$
Return $((i_0+1, fk_0), k_0, (X, \sigma))$

Algorithm $\mathsf{RKE.RKg}((hk, g, Y), y, (i_1, fk_1), (X, \sigma), k_1)$
$acc \leftarrow (\sigma = \mathsf{F.Ev}(fk_1, X))$
If not $acc$ then return $((i_1, fk_1), k_1, acc)$
$s \leftarrow \mathsf{H.Ev}(hk, i_1 \,\|\, \sigma \,\|\, X \,\|\, X^y)$
$k_1 \leftarrow s[1 \ldots \mathsf{RKE.kl}]$
$fk_1 \leftarrow s[\mathsf{RKE.kl}+1 \ldots \mathsf{RKE.kl}+\mathsf{F.kl}]$
Return $((i_1+1, fk_1), k_1, acc)$

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

Consider changing $\mathsf{RATCHET\text{-}KE}[\mathbb{G}, \mathsf{F}, \mathsf{H}]$ to have $\mathsf{H}(hk, \cdot)$ take inputs strings of the form $w = i \,\|\, g^{x_i y}$. This enables the following attack. Assume that an attacker compromises sender's keys $k_0$ and $fk_0$ at some time period $i$, and immediately uses the compromised authenticity to establish new keys $k^*$ and $fk^*$, shared between the attacker and the receiver. Now let $ephk_{i+1} = (g^{x_{i+1}}, \sigma_{i+1})$ be a valid ephemeral key, produced by the sender at a time period $i+1$. The attacker can construct a malicious ephemeral key $ephk^* = (g^{x_{i+1}}, \sigma^*)$, where $\sigma^* = \mathsf{F.Ev}(fk^*, g^{x_{i+1}})$, and send it to the receiver. The receiver would accept $ephk^*$ and use the output of $\mathsf{H.Ev}(hk, (i+1) \,\|\, g^{x_{i+1} y})$ as a new key material, resulting in the same keys as those generated by the sender along with the ephemeral key $ephk_{i+1}$. This is a clear violation of our policy that keys derived from modified ephmeral keys should not leak information about validly generated keys.

Security Theorem. We now present our theorem bounding the advantage of an adversary breaking the KIND-security of ratcheted encryption scheme $\mathsf{RKE} = \mathsf{RATCHET\text{-}KE}[\mathbb{G}, \mathsf{F}, \mathsf{H}]$ to the SUFCMA-security of $\mathsf{F}$ and the ODHE-security of $\mathbb{G}, \mathsf{H}$.

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

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

*Adversary $\mathcal{F}$ makes at most $q_{\text{RatSend}}$ queries to its* Tag *oracle and $q_{\text{RatRec}}$ queries to its* Verify *oracle. Adversary $\mathcal{O}_1$ makes at most $q_{\text{RatSend}}$ queries to its* Up *oracle, 2 queries to its* Ch *oracle, $q_{\text{Exp}}$ 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 $q_{\text{RatSend}}$ queries to its* Up *oracle, $q_{\text{RatSend}} + q_{\text{RatRec}}$ queries to its* Ch *oracle,$q_{\text{Exp}}$ queries to its* Exp *oracle, and $q_{\text{RatRec}} + q_{\text{Exp}}$ queries to its* Hash *oracle.*

Proving this theorem is highly non-trivial, requiring a careful eye for detail lest one fall into subtle

traps. 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 ephemeral keys that will be accepted by the receiver without exposing. Unfortunately, there is a subtle flaw with this proof technique. The adversary may attempt to create a forged *ephk* before it has decided whether or not to expose. In this case we need to check the validity of their forgery with a MAC key *fk*, before we know whether it should be random or 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.

**Proof of Theorem 4.1:** Consider the sequence of games shown in Fig. 7. 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 $\mathsf{RKE}$ inserted. Additionally, a flag $\mathsf{valid}$ has been added. This flag keeps track of whether the most recent ephemeral key was passed unchanged from the sender to the receiver and thus the keys $k_1$ and $fk_1$ should be indistinguishable from random to adversary $\mathcal{D}$. In this case, the adversary should not be able to create an ephemeral key *ephk* that is accepted by RatRec unless it calls Exp or forwards along the validly generated *ephk*. We prove this with a hybrid argument over the games $\mathrm{G}_{0,0}, \ldots, \mathrm{G}_{0,q_{\mathrm{RatSend}}+1}$. Game $\mathrm{G}_{0,j}$ assumes forgery attempts fail for the first $j$ keys, sets a $\mathsf{bad}$ 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 $\mathsf{bad}$ flag is set.

The above gives us

$$\Pr[\mathrm{G}_{0,0}] = \Pr[\mathrm{KIND}^{\mathcal{D}}_{\mathsf{RKE}}], \tag{2}$$

and for all $j \in \{0, \ldots, q_{\mathrm{RatSend}}\}$ we have

$$\Pr[\mathrm{G}^*_{0,j}] = \Pr[\mathrm{G}_{0,j+1}]. \tag{3}$$

Furthermore, for all $j \in \{1, \ldots, q_{\mathrm{RatSend}}\}$ games $\mathrm{G}_{0,j}$ and $\mathrm{G}^*_{0,j}$ are identical until $\mathsf{bad}$, so the fundamental lemma of game playing [9] gives:

$$\Pr[\mathrm{G}_{0,j}] - \Pr[\mathrm{G}^*_{0,j}] \leq \Pr[\mathsf{bad}^{\mathrm{G}^*_{0,j}}], \tag{4}$$

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

Now consider the relationship between games $\mathrm{G}^*_{0,j}$ and $\mathrm{I}_j$ (the latter also shown in Fig. 7). Game $\mathrm{I}_j$ is identical to $\mathrm{G}^*_{0,j}$, except that in $\mathrm{I}_j$ the output of hash function $\mathsf{H}$ is replaced with a uniformly random string whenever $i + 1 = j$.

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

$b \leftarrow\!\!\$\ \{0,1\}$ ; $i_0 \leftarrow 0$ ; $i_1 \leftarrow 0$ ; $\mathsf{valid} \leftarrow \mathsf{true}$ ; $rand \leftarrow\!\!\$\ \mathsf{H.Out}$
$k_0 \leftarrow\!\!\$\ \{0,1\}^{\mathsf{RKE.kl}}$ ; $k_1 \leftarrow k_0$ ; $fk_0 \leftarrow\!\!\$\ \{0,1\}^{\mathsf{F.kl}}$ ; $fk_1 \leftarrow fk_0$
$hk \leftarrow\!\!\$\ \{0,1\}^{\mathsf{H.kl}}$ ; $g \leftarrow\!\!\$\ \mathbb{G}^*$ ; $y \leftarrow\!\!\$\ \mathbb{Z}_p$ ; $lpk \leftarrow (hk, g, g^y)$
$b' \leftarrow\!\!\$\ \mathcal{D}^{\textsc{RatSend},\textsc{RatRec},\textsc{Exp},\textsc{ChSend},\textsc{ChRec}}(lpk)$ ; Return $(b' = b)$

RatSend
If $\mathsf{op}[i_0] = \perp$ then $\mathsf{op}[i_0] \leftarrow$ "ch"
$x \leftarrow\!\!\$\ \mathbb{Z}_p$ ; $\sigma \leftarrow \mathsf{F.Ev}(fk_0, g^x)$ ; $ephk \leftarrow (g^x, \sigma)$
$s \leftarrow \mathsf{H.Ev}(hk, i_0 \,\|\, \sigma \,\|\, g^x \,\|\, g^{xy})$
If $i_0 + 1 = j$ then $s \leftarrow rand$ // $\mathrm{I}_j$
$\mathsf{auth}[i_0] \leftarrow ephk$ ; $i_0 \leftarrow i_0 + 1$ ; $k_0 \leftarrow s[1 \ldots \mathsf{RKE.kl}]$
$fk_0 \leftarrow s[\mathsf{RKE.kl} + 1 \ldots \mathsf{RKE.kl} + \mathsf{F.kl}]$ ; Return $ephk$

RatRec($ephk$)
$(X, \sigma) \leftarrow ephk$
If $\mathsf{valid}$ and $(\mathsf{op}[i_1] \neq$ "exp") and $(ephk \neq \mathsf{auth}[i_1])$ then
  If $i_1 < j$ then return $\mathsf{false}$
  If $i_1 = j$ then
    If $\sigma \neq \mathsf{F.Ev}(fk_1, X)$ then return $\mathsf{false}$
    $\mathsf{bad} \leftarrow \mathsf{true}$
    Return $\mathsf{false}$ // $\mathrm{G}^*_{0,j}$,$\mathrm{I}_j$
If $\sigma \neq \mathsf{F.Ev}(fk_1, X)$ then return $\mathsf{false}$
If $\mathsf{op}[i_1] =$ "exp" then $\mathsf{restricted} \leftarrow \mathsf{true}$
If $ephk = \mathsf{auth}[i_1]$ then
  $\mathsf{valid} \leftarrow \mathsf{true}$ ; $\mathsf{restricted} \leftarrow \mathsf{false}$
Else
  $\mathsf{valid} \leftarrow \mathsf{false}$
$s \leftarrow \mathsf{H.Ev}(hk, i_1 \,\|\, \sigma \,\|\, X \,\|\, X^y)$
If $i_1 + 1 = j$ then $s \leftarrow rand$ // $\mathrm{I}_j$
$i_1 \leftarrow i_1 + 1$ ; $k_1 \leftarrow s[1 \ldots \mathsf{RKE.kl}]$
$fk_1 \leftarrow s[\mathsf{RKE.kl} + 1 \ldots \mathsf{RKE.kl} + \mathsf{F.kl}]$ ; Return $\mathsf{true}$

Exp
If $\mathsf{op}[i_0] =$ "ch" then return $\perp$
$\mathsf{op}[i_0] \leftarrow$ "exp" ; Return $(x, (i_0, fk_0), k_0)$

ChSend
If $\mathsf{op}[i_0] =$ "exp" then return $\perp$
$\mathsf{op}[i_0] \leftarrow$ "ch"
If $\mathsf{rkey}[i_0] = \perp$ then $\mathsf{rkey}[i_0] \leftarrow\!\!\$\ \{0,1\}^{\mathsf{RKE.kl}}$
If $b = 1$ then return $k_0$ else return $\mathsf{rkey}[i_0]$

ChRec
If $\mathsf{restricted}$ then return $k_1$
If $\mathsf{op}[i_1] =$ "exp" then return $\perp$
$\mathsf{op}[i_1] \leftarrow$ "ch"
If $\mathsf{rkey}[i_1] = \perp$ then $\mathsf{rkey}[i_1] \leftarrow\!\!\$\ \{0,1\}^{\mathsf{RKE.kl}}$
If $b = 1$ then return $k_1$ else return $\mathsf{rkey}[i_1]$

Figure 7: **Games $\mathrm{G}_{0,j}$, $\mathrm{G}^*_{0,j}$, $\mathrm{I}_j$ for proof of Theorem 4.1.**

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

$j' \leftarrow\!\!\$\ \{1, \ldots, q_{\text{RatSend}}\}$ ; $b \leftarrow\!\!\$\ \{0,1\}$ ; $b' \leftarrow 0$
$i_0 \leftarrow 0$ ; $i_1 \leftarrow 0$ ; $\mathsf{valid} \leftarrow \mathsf{true}$
$k_0 \leftarrow\!\!\$\ \{0,1\}^{\mathsf{RKE.kl}}$ ; $k_1 \leftarrow k_0$
$fk_0 \leftarrow\!\!\$\ \{0,1\}^{\mathsf{F.kl}}$ ; $fk_1 \leftarrow fk_0$ ; $lpk \leftarrow (hk, g, Y)$
$\mathcal{D}^{\text{RatSendSim},\text{RatRecSim},\text{ExpSim},\text{ChSendSim},\text{ChRecSim}}(lpk)$
Return $b'$

RatRecSim($ephk$)

$(X, \sigma) \leftarrow ephk$
$\mathsf{forge} \leftarrow ((\mathsf{op}[i_1] \neq \text{“exp”}) \wedge (ephk \neq \mathsf{auth}[i_1]))$
If $\mathsf{valid}$ and $\mathsf{forge}$ then
  If $i_1 < j'$ then return $\mathsf{false}$
  If $i_1 = j'$ then
    If $\sigma \neq \mathsf{F.Ev}(fk_1, X)$ then return $\mathsf{false}$
    $\mathsf{bad} \leftarrow \mathsf{true}$ ; $b' \leftarrow 1$ ; Return $\mathsf{false}$
If $\sigma \neq \mathsf{F.Ev}(fk_1, X)$ then return $\mathsf{false}$
If $\mathsf{op}[i_1] =$ “exp” then $\mathsf{restricted} \leftarrow \mathsf{true}$
If $ephk = \mathsf{auth}[i_1]$ then
  $\mathsf{valid} \leftarrow \mathsf{true}$ ; $\mathsf{restricted} \leftarrow \mathsf{false}$
Else
  $\mathsf{valid} \leftarrow \mathsf{false}$
If $i_1 + 1 \neq j'$ then $s \leftarrow \text{Hash}(i_1, \sigma \,\|\, X, X)$
Else $s \leftarrow \text{Ch}(\sigma \,\|\, X)$
$i_1 \leftarrow i_1 + 1$ ; $k_1 \leftarrow s[1 \ldots \mathsf{RKE.kl}]$
$fk_1 \leftarrow s[\mathsf{RKE.kl} + 1 \ldots \mathsf{RKE.kl} + \mathsf{F.kl}]$
Return $\mathsf{true}$

ExpSim

If $\mathsf{op}[i_0] =$ “ch” then return $\perp$
$\mathsf{op}[i_0] \leftarrow$ “exp” ; $x \leftarrow \text{Exp}$
Return $(x, (i_0, fk_0), k_0)$

RatSendSim

If $\mathsf{op}[i_0] = \perp$ then $\mathsf{op}[i_0] \leftarrow$ “ch”
$X \leftarrow \text{Up}$ ; $\sigma \leftarrow \mathsf{F.Ev}(fk_0, X)$
$ephk \leftarrow (X, \sigma)$
If $i_0 + 1 \neq j'$ then
  $s \leftarrow \text{Hash}(i_0, \sigma \,\|\, X, X)$
Else
  $s \leftarrow \text{Ch}(\sigma \,\|\, X)$
$\mathsf{auth}[i_0] \leftarrow ephk$ ; $i_0 \leftarrow i_0 + 1$
$k_0 \leftarrow s[1 \ldots \mathsf{RKE.kl}]$
$fk_0 \leftarrow s[\mathsf{RKE.kl} + 1 \ldots \mathsf{RKE.kl} + \mathsf{F.kl}]$
Return $ephk$

ChSendSim

If $\mathsf{op}[i_0] =$ “exp” then return $\perp$
$\mathsf{op}[i_0] \leftarrow$ “ch”
If $\mathsf{rkey}[i_0] = \perp$ then
  $\mathsf{rkey}[i_0] \leftarrow\!\!\$\ \{0,1\}^{\mathsf{RKE.kl}}$
If $b = 1$ then return $k_0$
Else return $\mathsf{rkey}[i_0]$

ChRecSim

If $\mathsf{restricted}$ then return $k_1$
If $\mathsf{op}[i_1] =$ “exp” then return $\perp$
$\mathsf{op}[i_1] \leftarrow$ “ch”
If $\mathsf{rkey}[i_1] = \perp$ then
  $\mathsf{rkey}[i_1] \leftarrow\!\!\$\ \{0,1\}^{\mathsf{RKE.kl}}$
If $b = 1$ then return $k_1$
Else return $\mathsf{rkey}[i_1]$

Figure 8: Adversary $\mathcal{O}_1$ for proof of Theorem 4.1.

Note that if $j = 0$ then games $\text{G}^*_{0,j}$ and $\text{I}_j$ are identical so $\Pr[\mathsf{bad}^{\text{G}^*_{0,0}}] = \Pr[\mathsf{bad}^{\text{I}_0}]$. For other values of $j$ we relate the probability that these games set $\mathsf{bad}$ to the advantage of the oracle Diffie-Hellman adversary $\mathcal{O}_1$ that is defined in Fig. 8. Let $b_{\mathsf{odhe}}$ denote the challenge bit in game $\text{ODHE}^{\mathcal{O}_1}_{\mathbb{G},\mathsf{H}}$, 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 $\text{G}^*_{0,j'}$ or $\text{I}_{j'}$ 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 $\mathsf{bad}$ from being set (such as calling Exp when $i_0 = j'$) then $\mathcal{O}_1$ no longer perfectly simulates the view of $\mathcal{D}$, but it is irrelevant for our analysis. Thus for all $j \in \{1, \ldots, q_{\text{RatSend}}\}$, we have

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

Combining the above for all possible values of $j$ (and the fact that $\Pr[\mathsf{bad}^{\text{G}^*_{0,0}}] = \Pr[\mathsf{bad}^{\text{I}_0}]$) gives

$$\mathsf{Adv}^{\mathsf{odhe}}_{\mathbb{G},\mathsf{H},\mathcal{O}_1} = \sum_{j=0}^{q_{\text{RatSend}}} \frac{\Pr[\mathsf{bad}^{\text{G}^*_{0,j}}]}{q_{\text{RatSend}}} - \sum_{j=0}^{q_{\text{RatSend}}} \frac{\Pr[\mathsf{bad}^{\text{I}_j}]}{q_{\text{RatSend}}}. \tag{5}$$

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

$j' \leftarrow\!\!\$\ \{0,\ldots,q_{\text{RatSend}}\}$ ; $b \leftarrow\!\!\$\ \{0,1\}$
$i_0 \leftarrow 0$ ; $i_1 \leftarrow 0$ ; $\mathsf{valid} \leftarrow \mathsf{true}$
$rand \leftarrow\!\!\$\ \mathsf{H.Out}$ ; $k_0 \leftarrow\!\!\$\ \{0,1\}^{\mathsf{RKE.kl}}$ ; $k_1 \leftarrow k_0$
$fk_0 \leftarrow\!\!\$\ \{0,1\}^{\mathsf{F.kl}}$ ; $fk_1 \leftarrow fk_0$ ; $hk \leftarrow\!\!\$\ \{0,1\}^{\mathsf{H.kl}}$
$g \leftarrow\!\!\$\ \mathbb{G}^*$ ; $y \leftarrow\!\!\$\ \mathbb{Z}_p$ ; $lpk \leftarrow (hk, g, g^y)$
$\mathcal{D}^{\text{RatSendSim,RatRecSim,ExpSim,ChSendSim,ChRecSim}}(lpk)$

RatRecSim($ephk$)

$(X, \sigma) \leftarrow ephk$
$\mathsf{forge} \leftarrow ((\mathsf{op}[i_1] \neq \text{“exp”}) \wedge (ephk \neq \mathsf{auth}[i_1]))$
If valid and forge then
  If $i_1 < j'$ then return false
  If $i_1 = j'$ then
    If not $\text{Verify}(X, \sigma)$ then return false
    bad $\leftarrow$ true
    Return false
If $(i_1 = j')$ then
  If not $\text{Verify}(X, \sigma)$ then return false
Else
  If $\sigma \neq \mathsf{F.Ev}(fk_1, X)$ then return false
If $\mathsf{op}[i_1] =$ “exp” then restricted $\leftarrow$ true
If $ephk = \mathsf{auth}[i_1]$ then
  valid $\leftarrow$ true ; restricted $\leftarrow$ false
Else
  valid $\leftarrow$ false
$s \leftarrow \mathsf{H.Ev}(hk, i_1 \,\|\, \sigma \,\|\, X \,\|\, X^y)$
If $i_1 + 1 = j$ then $s \leftarrow rand$
$i_1 \leftarrow i_1 + 1$ ; $k_1 \leftarrow s[1 \ldots \mathsf{RKE.kl}]$
$fk_1 \leftarrow s[\mathsf{RKE.kl} + 1 \ldots \mathsf{RKE.kl} + \mathsf{F.kl}]$
Return true

RatSendSim

If $\mathsf{op}[i_0] = \perp$ then $\mathsf{op}[i_0] \leftarrow$ “ch”
$x \leftarrow\!\!\$\ \mathbb{Z}_p$
If $i_0 = j'$ then $\sigma \leftarrow \text{Tag}(g^x)$
Else $\sigma \leftarrow \mathsf{F.Ev}(fk_0, g^x)$
$s \leftarrow \mathsf{H.Ev}(hk, i_0 \,\|\, \sigma \,\|\, g^x \,\|\, g^{xy})$
If $i_0 + 1 = j$ then $s \leftarrow rand$
$ephk \leftarrow (g^x, \sigma)$ ; $\mathsf{auth}[i_0] \leftarrow ephk$
$i_0 \leftarrow i_0 + 1$ ; $k_0 \leftarrow s[1 \ldots \mathsf{RKE.kl}]$
$fk_0 \leftarrow s[\mathsf{RKE.kl} + 1 \ldots \mathsf{RKE.kl} + \mathsf{F.kl}]$
Return $ephk$

ExpSim

If $\mathsf{op}[i_0] =$ “ch” then return $\perp$
$\mathsf{op}[i_0] \leftarrow$ “exp”
Return $(x, (i_0, fk_0), k_0)$

ChSendSim

If $\mathsf{op}[i_0] =$ “exp” then return $\perp$
$\mathsf{op}[i_0] \leftarrow$ “ch”
If $\mathsf{rkey}[i_0] = \perp$ then
  $\mathsf{rkey}[i_0] \leftarrow\!\!\$\ \{0,1\}^{\mathsf{RKE.kl}}$
If $b = 1$ then return $k_0$
Else return $\mathsf{rkey}[i_0]$

ChRecSim

If restricted then return $k_1$
If $\mathsf{op}[i_1] =$ “exp” then return $\perp$
$\mathsf{op}[i_1] \leftarrow$ “ch”
If $\mathsf{rkey}[i_1] = \perp$ then
  $\mathsf{rkey}[i_1] \leftarrow\!\!\$\ \{0,1\}^{\mathsf{RKE.kl}}$
If $b = 1$ then return $k_1$
Else return $\mathsf{rkey}[i_1]$

Figure 9: Adversary $\mathcal{F}$ for proof of Theorem 4.1.

To complete our hybrid argument, we can finally bound the probability that bad gets set true in $\text{I}_j$. Doing so requires adversary $\mathcal{D}$ to successfully forge a MAC tag for a uniformly random key, allowing us to reduce to the security of F. Formally, we use $\mathcal{D}$ to construct an adversary $\mathcal{F}$ attacking the SUFCMA security of F. Adversary $\mathcal{F}$ (shown in Fig. 9) simulates adversary $\mathcal{D}$ and guesses when it will first create a forgery. $\mathcal{F}$ simulates game $\text{I}_j$ for adversary $\mathcal{D}$ until that point, and uses it's own SUFCMA oracles to answer $\mathcal{D}$'s queries at the time when it expects the forgery. Similar to the earlier case when $\mathcal{O}_1$ simulated $\mathcal{D}$, adversary $\mathcal{F}$ may fail to simulate $\text{I}_j$ for adversary $\mathcal{D}$ when the latter performs certain actions that preclude bad from being set; this does not affect our analysis. Thus for $j \in \{0,\ldots,q_{\text{RatSend}}\}$,

$$\Pr[\mathsf{bad}^{\text{I}_j}] \leq \Pr[\,\text{SUFCMA}_{\mathsf{F}}^{\mathcal{F}} \mid j' = j\,]. \tag{6}$$

The above work allows us to transition to game $\text{G}_{0,q_{\text{RatSend}}+1}$. From there we will move to games $\text{G}_1, \text{G}_2$ shown in Fig. 10.

Games $\mathrm{G}_1$–$\mathrm{G}_2$

$b \leftarrow\!\!\$\ \{0,1\}$ ; $i_0 \leftarrow 0$ ; $i_1 \leftarrow 0$ ; $\mathsf{valid} \leftarrow \mathsf{true}$
$k_0[0] \leftarrow\!\!\$\ \{0,1\}^{\mathsf{RKE.kl}}$ ; $k_1 \leftarrow k_0[0]$ ; $fk_0[0] \leftarrow\!\!\$\ \{0,1\}^{\mathsf{F.kl}}$ ; $fk_1 \leftarrow fk_0[0]$
$hk \leftarrow\!\!\$\ \{0,1\}^{\mathsf{H.kl}}$ ; $g \leftarrow\!\!\$\ \mathbb{G}^*$ ; $y \leftarrow\!\!\$\ \mathbb{Z}_p$ ; $lpk \leftarrow (hk, g, g^y)$
$b' \leftarrow\!\!\$\ \mathcal{D}^{\textsc{RatSend},\textsc{RatRec},\textsc{Exp},\textsc{ChSend},\textsc{ChRec}}(lpk)$ ; Return $(b' = b)$

RatSend

If $\mathsf{op}[i_0] = \perp$ then $\mathsf{op}[i_0] \leftarrow$ "ch"
$x \leftarrow\!\!\$\ \mathbb{Z}_p$ ; $\sigma \leftarrow \mathsf{F.Ev}(fk_0[i_0], g^x)$ ; $ephk \leftarrow (g^x, \sigma)$
$s \leftarrow \mathsf{H.Ev}(hk, i_0 \,\|\, \sigma \,\|\, g^x \,\|\, g^{xy})$ // $\mathrm{G}_1$
$s \leftarrow\!\!\$\ \mathsf{H.Out}$ // $\mathrm{G}_2$
$\mathsf{auth}[i_0] \leftarrow ephk$ ; $i_0 \leftarrow i_0 + 1$ ; $k_0[i_0] \leftarrow s[1 \ldots \mathsf{RKE.kl}]$
$fk_0[i_0] \leftarrow s[\mathsf{RKE.kl} + 1 \ldots \mathsf{RKE.kl} + \mathsf{F.kl}]$ ; Return $ephk$

RatRec($ephk$)

$(X, \sigma) \leftarrow ephk$
If $\mathsf{valid}$ and $(\mathsf{op}[i_1] \neq$ "exp") and $(ephk \neq \mathsf{auth}[i_1])$ then
  Return $\mathsf{false}$
If $\mathsf{valid}$ then $fk_1 \leftarrow fk_0[i_1]$
If $(\sigma \neq \mathsf{F.Ev}(fk_1, X))$ then return $\mathsf{false}$
If $\mathsf{op}[i_1] =$ "exp" then $\mathsf{restricted} \leftarrow \mathsf{true}$
If $ephk = \mathsf{auth}[i_1]$
  $\mathsf{valid} \leftarrow \mathsf{true}$ ; $\mathsf{restricted} \leftarrow \mathsf{false}$ ; $i_1 \leftarrow i_1 + 1$
Else
  $\mathsf{valid} \leftarrow \mathsf{false}$
  $s \leftarrow \mathsf{H.Ev}(hk, i_1 \,\|\, \sigma \,\|\, X \,\|\, X^y)$
  $i_1 \leftarrow i_1 + 1$ ; $k_1 \leftarrow s[1 \ldots \mathsf{RKE.kl}]$
  $fk_1 \leftarrow s[\mathsf{RKE.kl} + 1 \ldots \mathsf{RKE.kl} + \mathsf{F.kl}]$
Return $\mathsf{true}$

Exp

If $\mathsf{op}[i_0] =$ "ch" then return $\perp$
$\mathsf{op}[i_0] \leftarrow$ "exp" ; $(X, \sigma) \leftarrow \mathsf{auth}[i_0 - 1]$
$s \leftarrow \mathsf{H.Ev}(hk, (i_0 - 1) \,\|\, \sigma \,\|\, X \,\|\, X^y)$ ; $k_0[i_0] \leftarrow s[1 \ldots \mathsf{RKE.kl}]$
$fk_0[i_0] \leftarrow s[\mathsf{RKE.kl} + 1 \ldots \mathsf{RKE.kl} + \mathsf{F.kl}]$
Return $(x, (i_0, fk_0[i_0]), k_0[i_0])$

ChSend

If $\mathsf{op}[i_0] =$ "exp" then return $\perp$
$\mathsf{op}[i_0] \leftarrow$ "ch"
If $\mathsf{rkey}[i_0] = \perp$ then $\mathsf{rkey}[i_0] \leftarrow\!\!\$\ \{0,1\}^{\mathsf{RKE.kl}}$
If $b = 1$ then return $k_0[i_0]$ else return $\mathsf{rkey}[i_0]$

ChRec

If $\mathsf{restricted}$ then return $k_1$
If $\mathsf{op}[i_1] =$ "exp" then return $\perp$
$\mathsf{op}[i_1] \leftarrow$ "ch"
If $\mathsf{rkey}[i_1] = \perp$ then $\mathsf{rkey}[i_1] \leftarrow\!\!\$\ \{0,1\}^{\mathsf{RKE.kl}}$
If $\mathsf{valid}$ then $k_1 \leftarrow k_0[i_1]$
If $b = 1$ then return $k_1$ else return $\mathsf{rkey}[i_1]$

Figure 10: **Games $\mathrm{G}_1$–$\mathrm{G}_2$ for proof of Theorem 4.1.**

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

$b \leftarrow\!\!\$ \{0,1\}$ ; $i_0 \leftarrow 0$ ; $i_1 \leftarrow 0$ ; $\mathsf{valid} \leftarrow \mathsf{true}$
$k_0[0] \leftarrow\!\!\$ \{0,1\}^{\mathsf{RKE.kl}}$ ; $k_1 \leftarrow k_0[0]$
$fk_0[0] \leftarrow\!\!\$ \{0,1\}^{\mathsf{F.kl}}$ ; $fk_1 \leftarrow fk_0[0]$ ; $hk \leftarrow\!\!\$ \{0,1\}^{\mathsf{H.kl}}$
$g \leftarrow\!\!\$ \mathbb{G}^*$ ; $y \leftarrow\!\!\$ \mathbb{Z}_p$ ; $lpk \leftarrow (hk, g, Y)$ ;
$b' \leftarrow\!\!\$ \mathcal{D}^{\text{RatSendSim,RatRecSim,ExpSim,ChSendSim,ChRecSim}}(lpk)$
If $(b' = b)$ then return 1 else return 0

RatSendSim

If $\mathsf{op}[i_0] = \perp$ then
  $\mathsf{op}[i_0] \leftarrow$ "ch"
  If $i_0 \neq 0$ then
    $(X, \sigma) \leftarrow \mathsf{auth}[i_0 - 1]$ ; $s \leftarrow \text{Ch}(\sigma||X)$
    SaveKeys$(i_0, s)$
$X \leftarrow \text{Up}$ ; $\sigma \leftarrow \mathsf{F.Ev}(fk_0[i_0], X)$ ; $ephk \leftarrow (X, \sigma)$
$\mathsf{auth}[i_0] \leftarrow ephk$ ; $i_0 \leftarrow i_0 + 1$ ; Return $ephk$

RatRecSim$(ephk)$

$(X, \sigma) \leftarrow ephk$
$\mathsf{forge} \leftarrow ((\mathsf{op}[i_1] \neq \text{"exp"}) \wedge (ephk \neq \mathsf{auth}[i_1]))$
If $\mathsf{valid}$ and $\mathsf{forge}$ then return $\mathsf{false}$
If $\mathsf{valid}$ then $fk_1 \leftarrow fk_0[i_1]$
If $(\sigma \neq \mathsf{F.Ev}(fk_1, X))$ then return $\mathsf{false}$
If $\mathsf{op}[i_1] =$ "exp" then $\mathsf{restricted} \leftarrow \mathsf{true}$
If $ephk = \mathsf{auth}[i_1]$
  $\mathsf{valid} \leftarrow \mathsf{true}$ ; $\mathsf{restricted} \leftarrow \mathsf{false}$ ; $i_1 \leftarrow i_1 + 1$
Else
  $\mathsf{valid} \leftarrow \mathsf{false}$ ; $s \leftarrow \text{Hash}(i_1, \sigma||X, X)$
  $i_1 \leftarrow i_1 + 1$ ; $k_1 \leftarrow s[1 \ldots \mathsf{RKE.kl}]$
  $fk_1 \leftarrow s[\mathsf{RKE.kl} + 1 \ldots \mathsf{RKE.kl} + \mathsf{F.kl}]$
Return $\mathsf{true}$

SaveKeys$(i, s)$

$k_0[i] \leftarrow s[1 \ldots \mathsf{RKE.kl}]$
$fk_0[i] \leftarrow s[\mathsf{RKE.kl} + 1 \ldots \mathsf{RKE.kl} + \mathsf{F.kl}]$

ExpSim

If $\mathsf{op}[i_0] =$ "ch" then return $\perp$
If $\mathsf{op}[i_0] = \perp \wedge i_0 \neq 0$ then
  $x \leftarrow \text{Exp}$
  $(X, \sigma) \leftarrow \mathsf{auth}[i_0 - 1]$
  $s \leftarrow \text{Hash}(i_0 - 1, \sigma||X, X)$
  SaveKeys$(i_0, s)$
$\mathsf{op}[i_0] \leftarrow$ "exp"
Return $(x, (i_0, fk_0[i_0]), k_0[i_0])$

ChSendSim

If $\mathsf{op}[i_0] =$ "exp" then return $\perp$
If $\mathsf{op}[i_0] = \perp \wedge i_0 \neq 0$ then
  $(X, \sigma) \leftarrow \mathsf{auth}[i_0 - 1]$
  $s \leftarrow \text{Ch}(\sigma||X)$
  SaveKeys$(i_0, s)$
$\mathsf{op}[i_0] \leftarrow$ "ch"
If $\mathsf{rkey}[i_0] = \perp$ then
  $\mathsf{rkey}[i_0] \leftarrow\!\!\$ \{0,1\}^{\mathsf{RKE.kl}}$
If $b = 1$ then return $k_0[i_0]$
Else return $\mathsf{rkey}[i_0]$

ChRecSim

If $\mathsf{restricted}$ then return $k_1$
If $\mathsf{op}[i_1] =$ "exp" then return $\perp$
If $\mathsf{op}[i_1] = \perp \wedge i_1 \neq 0$ then
  $(X, \sigma) \leftarrow \mathsf{auth}[i_1 - 1]$
  $s \leftarrow \text{Ch}(\sigma||X)$
  SaveKeys$(i_1, s)$
$\mathsf{op}[i_1] \leftarrow$ "ch"
If $\mathsf{rkey}[i_1] = \perp$ then
  $\mathsf{rkey}[i_1] \leftarrow\!\!\$ \{0,1\}^{\mathsf{RKE.kl}}$
If $\mathsf{valid}$ then $k_1 \leftarrow k_0[i_1]$
If $b = 1$ then return $k_1$
Else return $\mathsf{rkey}[i_1]$

Figure 11: Adversary $\mathcal{O}_2$ for proof of Theorem 4.1.

Game $\text{G}_1$ is identical to $\text{G}_{0,q_{\text{RatSend}}+1}$, but has been rewritten to allow the final game transition of our proof. The complicated, nested if-condition at the beginning of RatRec has been simplified because $i_1 < q_{\text{RatSend}} + 1$ always holds when $\mathsf{valid}$ is true. Additionally, when $\mathsf{valid}$ is true (and thus a valid $ephk$ has been forwarded between RatSend and RatRec) we delay setting $k_1, fk_1$ until they are about to be used, at which point they are set to match the appropriate $k_0, fk_0$ that have been stored in a table. We have

$$\Pr[\text{G}_{0,q_{\text{RatSend}}+1}] = \Pr[\text{G}_1]. \tag{7}$$

Games $\Pr[\text{G}_1]$ and $\Pr[\text{G}_2]$ differ only in that, in $\text{G}_2$, values of $k_0$ and $fk_0$ are chosen at random instead of as the output of $\mathsf{H}$ (unless Exp is called in which case we reset them to the correct output of $\mathsf{H}$). We bound the difference between $\Pr[\text{G}_1]$ and $\Pr[\text{G}_2]$ by the advantage of the Diffie-Hellman adversary $\mathcal{O}_2$ that is defined in Fig. 11. Let $b_{\mathsf{odhe}}$ denote the challenge bit in game $\text{ODHE}^{\mathcal{O}_2}_{\mathbb{G},\mathsf{H}}$, and let $b'$ denote the corresponding guess made by the adversary $\mathcal{O}_2$. $\mathcal{O}_2$ perfectly simulates the view

of $\mathcal{D}$ in $\mathrm{G}_1$ when $b_{\mathsf{odhe}} = 1$, and it perfectly simulates the view of $\mathrm{G}_2$ when $b_{\mathsf{odhe}} = 0$. Thus,

$$\Pr[\mathrm{G}_1] = \Pr[\,b' = 1 \,|\, b_{\mathsf{odhe}} = 1\,] \qquad \text{and} \qquad \Pr[\mathrm{G}_2] = \Pr[\,b' = 1 \,|\, b_{\mathsf{odhe}} = 0\,].$$

It follows that

$$\mathsf{Adv}^{\mathsf{odhe}}_{\mathbb{G},\mathsf{H},\mathcal{O}_2} = \Pr[\mathrm{G}_1] - \Pr[\mathrm{G}_2]. \tag{8}$$

As a result of the above equations, we get:

$$\begin{aligned}
\Pr[\mathrm{KIND}^{\mathcal{D}}_{\mathsf{RKE}}] &= \Pr[\mathrm{G}_{0,0}] = \Pr[\mathrm{G}_1] + \sum_{j=0}^{q_{\textsc{RatSend}}} \Pr[\mathsf{bad}^{\mathrm{G}^*_{0,j}}] \\
&= \Pr[\mathrm{G}_1] + q_{\textsc{RatSend}} \cdot \mathsf{Adv}^{\mathsf{odhe}}_{\mathbb{G},\mathsf{H},\mathcal{O}_1} + \sum_{j=0}^{q_{\textsc{RatSend}}} \Pr[\mathsf{bad}^{\mathrm{I}_j}] \\
&\leq q_{\textsc{RatSend}} \cdot \mathsf{Adv}^{\mathsf{odhe}}_{\mathbb{G},\mathsf{H},\mathcal{O}_1} + (q_{\textsc{RatSend}} + 1) \cdot \mathsf{Adv}^{\mathsf{sufcma}}_{\mathsf{F},\mathcal{F}} + \Pr[\mathrm{G}_1] \\
&= q_{\textsc{RatSend}} \cdot \mathsf{Adv}^{\mathsf{odhe}}_{\mathbb{G},\mathsf{H},\mathcal{O}_1} + (q_{\textsc{RatSend}} + 1) \cdot \mathsf{Adv}^{\mathsf{sufcma}}_{\mathsf{F},\mathcal{F}} + \mathsf{Adv}^{\mathsf{odhe}}_{\mathbb{G},\mathsf{H},\mathcal{O}_2} + \Pr[\mathrm{G}_2].
\end{aligned}$$

Finally, $\Pr[\mathrm{G}_2] = \frac{1}{2}$ because the view of $\mathcal{D}$ is independent of $b$ in $\mathrm{G}_2$. To see this, first note that oracle ChSend returns uniformly random bits regardless of the challenge bit. So we only need to verify that the ChRec returns the same random bits if its last if-statement is reached. This could only fail to occur if ChRec was called when restricted and valid are both false. However, flags restricted and valid can only be simultaneously false at the end of an oracle call to RatRec if they were already false at the time when this oracle was called. ▌

# 5 Ratcheted encryption

We now define ratcheted encryption schemes, and show how to construct one by composing 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. We define a security notion for ratcheted encryption and reduce the security of our construction to the security of the underlying schemes.

The results in this section serve as an example of building new schemes by composing ratcheted key exchange with other primitives. There is a lot of room for capturing different security goals and building the corresponding schemes. One of the important goals is to model the *Double Ratchet Algorithm* [3] used in multiple real-world messaging applications, such as in WhatsApp [22] and in the Secret Conversations mode of Facebook Messenger [17]. This would require to introduce a second layer of key ratcheting, which can be possibly done by using the output keys of ratcheted key exchange to initialize a *forward-secure* symmetric encryption scheme. We currently do not capture this possibility; both the syntax and the security definitions we use would need to be extended for this purpose.

Ratcheted encryption schemes. Our definition of ratcheted encryption schemes extends the definition of ratcheted key exchange schemes to add an encryption algorithm RE.Enc and a decryption algorithm RE.Dec. Likewise, the correctness property that we require from ratcheted encryption schemes extends the correctness property of ratcheted key exchange schemes to add the requirement that any messages encrypted using sender's key should be correctly decryptable using the

Algorithm $\mathsf{RE.IKg}$
$(k, sek_0, (lpk, lsk, sek_1)) \leftarrow\!\!\$\ \mathsf{RKE.IKg}$
Return $(k, sek_0, (lpk, lsk, sek_1))$

Algorithm $\mathsf{RE.SKg}(lpk, sek_0; r)$
$(sek_0, k_0, ephk) \leftarrow \mathsf{RKE.SKg}(lpk, sek_0; r)$
Return $(sek_0, k_0, ephk)$

Algorithm $\mathsf{RE.RKg}(lpk, lsk, sek_1, ephk, k_1)$
$(sek_1, k_1, acc) \leftarrow\!\!\$\ \mathsf{RKE.RKg}(lpk, lsk, sek_1, ephk, k_1)$
Return $(sek_1, k_1, acc)$

Algorithm $\mathsf{RE.Enc}(k_0, n, m, h)$
$c \leftarrow \mathsf{SE.Enc}(k_0, n, m, h)$
Return $c$

Algorithm $\mathsf{RE.Dec}(k_1, n, c, h)$
$m \leftarrow \mathsf{SE.Dec}(k_1, n, c, h)$
Return $m$

Figure 12: Ratcheted encryption scheme $\mathsf{RE} = \mathsf{RATCHET\text{-}ENC}[\mathsf{RKE}, \mathsf{SE}]$.

corresponding receiver's key.

A ratcheted encryption scheme RE specifies algorithms RE.IKg, RE.SKg, RE.RKg, RE.Enc and RE.Dec, where RE.Enc and RE.Dec are deterministic. Associated to RE is a nonce space RE.NS, sender's key generation randomness space RE.RS, and a ciphertext length function $\mathsf{RE.cl}\colon \mathbb{N} \to \mathbb{N}$. Initial key generation algorithm RE.IKg returns $k, sek_0, (lpk, lsk, sek_1)$, where $k$ is a (symmetric) encryption key, $sek_0$ is a sender's session key, and $lpk, lsk, sek_1$ are receiver's long-term public key, receiver's long-term secret key, and receiver's session key, respectively. Sender's key generation algorithm RE.SKg takes $lpk, sek_0$ and randomness $r \in \mathsf{RE.RS}$ to return a new sender's session key $sek_0$, a new sender's encryption key $k_0$, and an ephemeral key $ephk$. Receiver's key generation algorithm RE.RKg takes $lpk, lsk, sek_1, ephk$ and receiver's encryption key $k_1$ to return a new receiver's session key $sek_1$, a new receiver's encryption key $k_1$, and a flag $acc \in \{\mathsf{true}, \mathsf{false}\}$. Encryption algorithm RE.Enc takes $k_0$, a nonce $n \in \mathsf{RE.NS}$, a plaintext message $m \in \{0,1\}^*$ and a header $h \in \{0,1\}^*$ to return a ciphertext $c \in \{0,1\}^{\mathsf{RE.cl}(|m|)}$. Decryption algorithm RE.Dec takes $k_1, n, c, h$ to return $m \in \{0,1\}^* \cup \{\perp\}$.

Consider game COR of Fig. 4 associated to a ratcheted encryption scheme RATCHET 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 RATCHET is defined as $\mathsf{Adv}^{\mathsf{cor}}_{\mathsf{RATCHET},\mathcal{C}} = 1 - \Pr[\mathrm{COR}^{\mathcal{C}}_{\mathsf{RATCHET}}]$. Correctness property requires that $\mathsf{Adv}^{\mathsf{cor}}_{\mathsf{RATCHET},\mathcal{C}} = 0$ for all unbounded adversaries $\mathcal{C}$.

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 of the latter with oracles Enc and Dec. Oracles RatSend, RatRec, Exp are the same in both games. Oracles Enc and Dec are defined to allow functionality that is similar to the standard definition of nonce-based authenticated encryption, the main difference being that Enc uses *the sender's key* to encrypt messages and Dec uses *the receiver's key* to decrypt messages. 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 these two cases.

We note that the adversary is only allowed to get a single encryption for each unique pair of $(i_0, n)$. 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 $(i_0, n, m)$ (or even $(i_0, n, m, h)$) are repeated, but it would increasingly limit the choice of the underlying

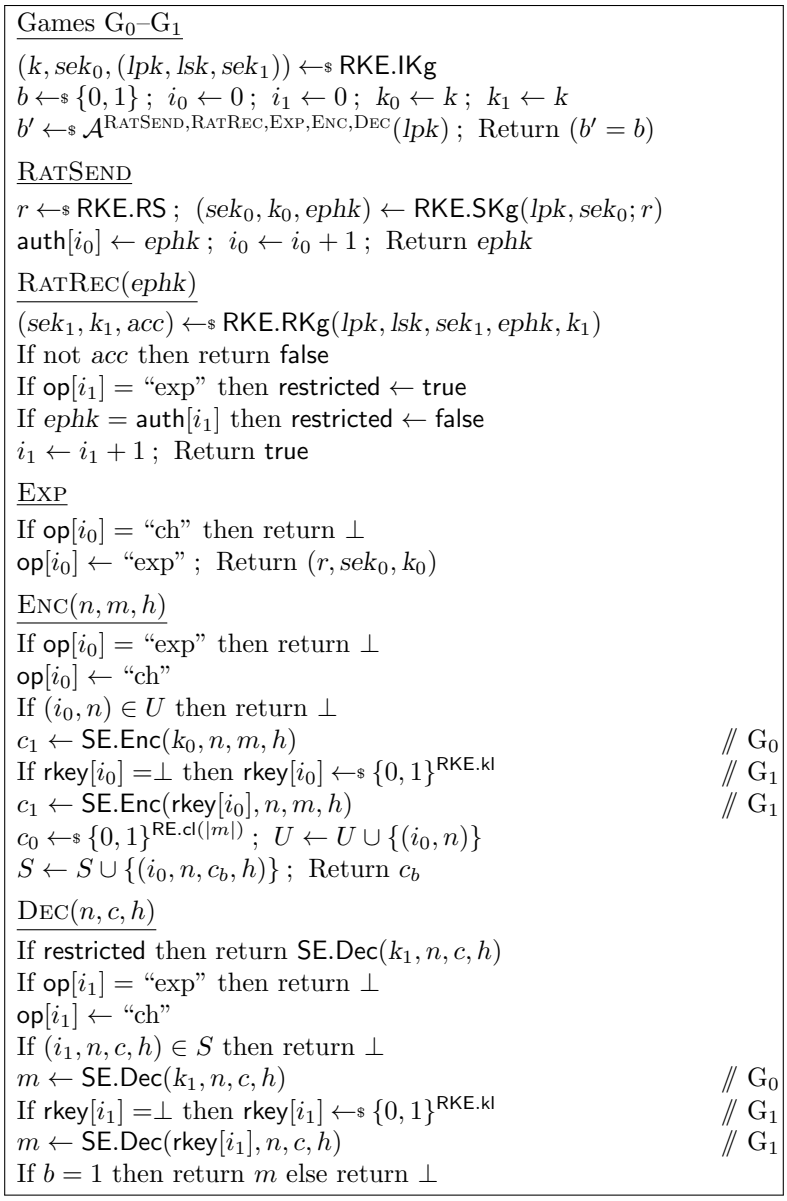


Figure 13: **Games for proof of Theorem 5.1.**

symmetric schemes that can be used for this purpose (fewer schemes would satisfy stronger security definitions of multi-user authenticated encryption). The advantage of $\mathcal{A}$ in breaking the RAE security of RE is defined as $\mathsf{Adv}^{\mathsf{rae}}_{\mathsf{RE},\mathcal{A}} = 2\Pr[\mathrm{RAE}^{\mathcal{A}}_{\mathsf{RE}}] - 1$.

Ratcheted encryption scheme RATCHET-ENC. Let RKE be a ratcheted key exchange scheme. Let SE be a symmetric encryption scheme such that SE.kl = RKE.kl. We build a ratcheted encryption scheme RE = RATCHET-ENC[RKE, SE] as defined in Fig. 12, with RE.NS = SE.NS, RE.RS = RKE.RS and RE.cl = SE.cl.

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

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

$b \leftarrow\!\!\$\ \{0,1\}$ ; $i_0 \leftarrow 0$ ; $i_1 \leftarrow 0$
$b' \leftarrow\!\!\$\ \mathcal{A}^{\text{RatSendSim},\text{RatRecSim},\text{ExpSim},\text{EncSim},\text{DecSim}}(lpk)$
If $(b' = b)$ then return 1 else return 0

EncSim$(n, m, h)$

If $\mathsf{op}[i_0]$ = "exp" then return $\perp$
$\mathsf{op}[i_0] \leftarrow$ "ch"
If $(i_0, n) \in U$ then return $\perp$
$k_0 \leftarrow\!\!\$$ ChSend ; $c_1 \leftarrow \mathsf{SE.Enc}(k_0, n, m, h)$
$c_0 \leftarrow\!\!\$\ \{0,1\}^{\mathsf{RE.cl}(|m|)}$ ; $U \leftarrow U \cup \{(i_0, n)\}$
$S \leftarrow S \cup \{(i_0, n, c_b, h)\}$ ; Return $c_b$

DecSim$(n, c, h)$

If $\mathsf{restricted}$ then
  $k_1 \leftarrow\!\!\$$ ChRec ; Return $\mathsf{SE.Dec}(k_1, n, c, h)$
If $\mathsf{op}[i_1]$ = "exp" then return $\perp$
$\mathsf{op}[i_1] \leftarrow$ "ch"
If $(i_1, n, c, h) \in S$ then return $\perp$
$k_1 \leftarrow\!\!\$$ ChRec ; $m \leftarrow \mathsf{SE.Dec}(k_1, n, c, h)$
If $b = 1$ then return $m$ else return $\perp$

RatSendSim

$ephk \leftarrow\!\!\$$ RatSend
$\mathsf{auth}[i_0] \leftarrow ephk$
$i_0 \leftarrow i_0 + 1$
Return $ephk$

RatRecSim$(ephk)$

$\mathsf{success} \leftarrow\!\!\$$ RatRec$(ephk)$
If $\mathsf{success}$ then
  $u_0 \leftarrow (\mathsf{op}[i_1]$ = "exp")
  $u_1 \leftarrow (ephk = \mathsf{auth}[i_1])$
  If $u_0$ then $\mathsf{restricted} \leftarrow \mathsf{true}$
  If $u_1$ then $\mathsf{restricted} \leftarrow \mathsf{false}$
  $i_1 \leftarrow i_1 + 1$
Return $\mathsf{success}$

ExpSim

If $\mathsf{op}[i_0]$ = "ch" then return $\perp$
$\mathsf{op}[i_0] \leftarrow$ "exp"
Return Exp

Figure 14: Adversary $\mathcal{D}$ for proof of Theorem 5.1.

*queries to its* RatRec *oracle,* $q_{\text{Exp}}$ *queries to its* Exp *oracle,* $q_{\text{Enc}}$ *queries to its* Enc *oracle, and* $q_{\text{Dec}}$ *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* $q_{\text{Exp}}$ *queries to its* Exp *oracle,* $q_{\text{Enc}}$ *queries to its* ChSend *oracle,* $q_{\text{Dec}}$ *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,* $q_{\text{Enc}}$ *queries to its* Enc *oracle, and* $q_{\text{Dec}}$ *queries to its* Dec *oracle.*

**Proof of Theorem 5.1:** Consider games $\text{G}_0, \text{G}_1$ of Fig. 13. Lines not annotated with comments are common to both games. Game $\text{G}_0$ is equivalent to $\text{RAE}^{\mathcal{A}}_{\mathsf{RE}}$, so

$$\mathsf{Adv}^{\mathsf{rae}}_{\mathsf{RE},\mathcal{A}} = 2\Pr[\text{G}_0] - 1. \tag{9}$$

Game $\text{G}_1$ differs from game $\text{G}_0$ by using uniformly random keys to answer Enc and Dec oracle queries. Both games use real keys to answer Exp oracle queries.

First, we construct an adversary $\mathcal{D}$ against the KIND-security of RKE, as defined in Fig. 14. Adversary $\mathcal{D}$ simulates adversary $\mathcal{A}$ as follows. $\mathcal{A}$'s oracle queries to RatSend, RatRec and Exp are directly answered by the corresponding $\mathcal{D}$'s oracles (but $\mathcal{D}$ also does some bookkeeping to maintain the states that are necessary for simulating other oracle queries). $\mathcal{D}$ simulates $\mathcal{A}$'s queries to Enc and Dec by calling it's own oracles ChSend and ChRec and using the received challenge keys to encrypt and decrypt the messages itself. Let $b$ denote the challenge bit in game $\text{KIND}^{\mathcal{D}}_{\mathsf{RKE}}$, and let $b'$ denote the corresponding guess made by the adversary $\mathcal{D}$. We have $\Pr[\text{G}_0] =$

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

$(k, sek_0, (lpk, lsk, sek_1)) \leftarrow\!\!\$\ \mathsf{RKE.IKg}$
$v \leftarrow 0\,;\ i_0 \leftarrow 0\,;\ i_1 \leftarrow 0\,;\ k_0 \leftarrow k\,;\ k_1 \leftarrow k$
$b' \leftarrow\!\!\$\ \mathcal{A}^{\text{RatSendSim},\text{RatRecSim},\text{ExpSim},\text{EncSim},\text{DecSim}}(lpk)$
Return $b'$

RatSendSim

$r \leftarrow\!\!\$\ \mathsf{RKE.RS}\,;\ z \leftarrow \mathsf{RKE.SKg}(lpk, sek_0; r)$
$(sek_0, k_0, ephk) \leftarrow z\,;\ \mathsf{auth}[i_0] \leftarrow ephk\,;\ i_0 \leftarrow i_0 + 1$
While $v < i_0$ do
  New ; $v \leftarrow v + 1$
Return $ephk$

RatRecSim($ephk$)

$(sek_1, k_1, acc) \leftarrow\!\!\$\ \mathsf{RKE.RKg}(lpk, lsk, sek_1, ephk, k_1)$
If not $acc$ then return false
If $\mathsf{op}[i_1]$ = "exp" then $\mathsf{restricted} \leftarrow \mathsf{true}$
If $ephk = \mathsf{auth}[i_1]$ then $\mathsf{restricted} \leftarrow \mathsf{false}$
$i_1 \leftarrow i_1 + 1$
While $v < i_1$ do
  New ; $v \leftarrow v + 1$
Return true

ExpSim

If $\mathsf{op}[i_0]$ = "ch" then return $\perp$
$\mathsf{op}[i_0] \leftarrow$ "exp"
Return $(r, sek_0, k_0)$

Enc$(n, m, h)$

If $\mathsf{op}[i_0]$ = "exp" then return $\perp$
$\mathsf{op}[i_0] \leftarrow$ "ch"
Return Enc$(i_0, n, m, h)$

Dec$(n, c, h)$

If restricted then
  Return $\mathsf{SE.Dec}(k_1, n, c, h)$
If $\mathsf{op}[i_1]$ = "exp" then return $\perp$
$\mathsf{op}[i_1] \leftarrow$ "ch"
Return Dec$(i_1, n, c, h)$

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

$\Pr[\,b' = 1 \,|\, b = 1\,]$ and $\Pr[\mathrm{G}_1] = \Pr[\,b' = 1 \,|\, b = 0\,]$. It follows that

$$\Pr[\mathrm{G}_0] - \Pr[\mathrm{G}_1] = \mathsf{Adv}^{\mathsf{kind}}_{\mathsf{RKE},\mathcal{D}}. \tag{10}$$

Next, we construct an adversary $\mathcal{N}$ against the MAE-security of SE, as defined in Fig. 15. Adversary $\mathcal{N}$ generates its own keys for the ratcheted key exchange scheme RKE, and uses them to answer $\mathcal{A}$'s queries to oracles RatSend, RatRec and Exp (as well as $\mathcal{A}$'s queries to Dec in the case when restricted is true). Furthermore, $\mathcal{A}$'s calls to Enc and Dec are answered using the corresponding oracles that are provided to $\mathcal{N}$ in game MAE. We have $\Pr[\mathrm{G}_1] = \mathrm{MAE}^{\mathcal{N}}_{\mathsf{SE}}$, so

$$\mathsf{Adv}^{\mathsf{mae}}_{\mathsf{SE},\mathcal{N}} = 2\Pr[\mathrm{G}_1] - 1. \tag{11}$$

The theorem statement follows from equations (9)–(11). ∎

# References


[1] The best encrypted messaging apps you can (and should) use today. https://heimdalsecurity.com/blog/the-best-encrypted-messaging-apps/. Accessed: 2016-09-15. 2

[2] Double ratchet algorithm (full specification of the protocol). https://github.com/trevp/double_ratchet/wiki. Accessed: 2016-09-15. 5

[3] Advanced cryptographic ratcheting. https://whispersystems.org/blog/advanced-ratcheting/, Nov. 26, 2013. 3, 5, 21

[4] M. Abdalla, M. Bellare, and P. Rogaway. The oracle Diffie-Hellman assumptions and an analysis of DHIES. In D. Naccache, editor, *CT-RSA 2001*, volume 2020 of *LNCS*, pages 143–158, San Francisco, CA, USA, Apr. 8–12, 2001. Springer, Heidelberg, Germany. 3, 4

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

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

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

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

[9] M. Bellare and P. Rogaway. The security of triple encryption and a framework for code-based game-playing proofs. In S. Vaudenay, editor, *EUROCRYPT 2006*, volume 4004 of *LNCS*, pages 409–426, St. Petersburg, Russia, May 28 – June 1, 2006. Springer, Heidelberg, Germany. 6, 15

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

[11] M. Bellare and B. S. Yee. Forward-security in private-key cryptography. In M. Joye, editor, *CT-RSA 2003*, volume 2612 of *LNCS*, pages 1–18, San Francisco, CA, USA, Apr. 13–17, 2003. Springer, Heidelberg, Germany. 5

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

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

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

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

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

[17] Messenger secret conversations. Technical Whitepaper, July 8, 2016. `https://fbnewsroomus.files.wordpress.com/2016/07/secret_conversations_whitepaper-1.pdf`. 21

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

[19] P. Rogaway. Authenticated-encryption with associated-data. In V. Atluri, editor, *ACM CCS 02*, pages 98–107, Washington D.C., USA, Nov. 18–22, 2002. ACM Press. 3, 4, 5, 7

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

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

[22] Whatsapp encryption overview. Technical white paper, Apr. 4, 2016. `https://www.whatsapp.com/security/WhatsApp-Security-Whitepaper.pdf`. 5, 21