Bidirectional Asynchronous Ratcheted Key Agreement with Linear Complexity

F. Betül Durak[1] and Serge Vaudenay[2](✉)

[1] Robert Bosch LLC - Research and Technology Center, Pittsburgh, USA
[2] Ecole Polytechnique Fédérale de Lausanne (EPFL), Lausanne, Switzerland
`serge.vaudenay@epfl.ch`

Abstract. Following up mass surveillance and privacy issues, modern secure communication protocols now seek more security such as forward secrecy and post-compromise security. They cannot rely on an assumption such as synchronization, predictable sender/receiver roles, or online availability. Ratcheting was introduced to address forward secrecy and post-compromise security in real-world messaging protocols. At CSF 2016 and CRYPTO 2017, ratcheting was studied either without zero round-trip time (0-RTT) or without bidirectional communication. At CRYPTO 2018, ratcheting with bidirectional communication was done using heavy key-update primitives. At EUROCRYPT 2019, another protocol was proposed. All those protocols use random oracles. Furthermore, exchanging n messages has complexity $\mathcal{O}(n^2)$ in general.

In this work, we define the bidirectional asynchronous ratcheted key agreement (BARK) with formal security notions. We provide a simple security model and design a secure BARK scheme using no key-update primitives, no random oracle, an with $\mathcal{O}(n)$ complexity. It is based on a public-key cryptosystem, a signature scheme, one-time symmetric encryption, and a collision-resistant hash function family. We further show that BARK (even unidirectional) implies public-key cryptography, meaning that it cannot solely rely on symmetric cryptography.

1 Introduction

In standard communication systems, protocols are designed to provide messaging services with end-to-end encryption. Essentially, secure communication reduces to continuously exchanging keys, because each message requires a new key. In bidirectional two-party secure communication, participants alternate their role as *senders* and *receivers*. The modern instant messaging protocols are substantially *asynchronous*. In other words, for a two-party communication, the messages should be transmitted (or the key exchange should be done) even though the counterpart is not online. Moreover, to be able to send the payload data without requiring online exchanges is a major design goal called *zero round trip time (0-RTT)*. Finally, the moment when a participant wants to send a message

---

A full version of this paper is available as eprint 2018/889 [8].

N. Attrapadung and T. Yagi (Eds.): IWSEC 2019, LNCS 11689, pp. 343–362, 2019.
https://doi.org/10.1007/978-3-030-26834-3_20

is undefined, meaning that participants use *random roles* (sender or receiver) without any synchronization. They could send messages at the same time.

Even though many systems were designed for the privacy of their users, they rapidly faced security vulnerabilities caused by the *compromises* of the participants' states. In this work, compromising a participant means to obtain some information about its internal state. We will call it *exposure*. The desired security notion is that compromised information should not uncover more than possible by trivial attacks. For instance, the compromised state of participants should not allow decryption of messages exchanged in the past. This is called *forward secrecy*. Typically, forward secrecy is obtained by updating states with a one-way function $\mathsf{x} \rightarrow \mathsf{H}(\mathsf{x}) \rightarrow \mathsf{H}(\mathsf{H}(\mathsf{x})) \rightarrow \ldots$ and deleting old entries [13,14]. A popular technique in mechanics, that allows forward movement but prevents moving backward is the use of a device called *ratchet*. In the context of secure communication, a ratchet-like action is achieved by using randomness in every state update so that a compromised state is not sufficient for the decryption of any future communication either. This is called *future secrecy* or *backward secrecy* or *post-compromise security* or even *self-healing*. One thesis of the present work is that healing after an active attack involving a forgery is not a nice property. We show that it implies insecurity. After one participant is compromised and impersonated, if communication self-heals, it means that some adversary can make a trivial attack which is not detected. We also show other insecurity cases. Hence, we rather mandate communication to be cut after active attacks.

Previous Work. The security of key exchange was studied by many authors. The prominent models are the CK and eCK models [4,12].

Techniques for ratcheting first appeared in real life protocols. It appeared in the Off-the-Record (OTR) communication system by Borisov et al. [3]. The Signal protocol designed by Open Whisper Systems [16] later gained a lot of interest from message communication companies. Today, the WhatsApp messaging application reached billions of users worldwide [18]. It uses the Signal protocol. A broad survey about various techniques and terminologies was made at S&P 2015 by Unger et al. [17]. At CSF 2016, Cohn-Gordon et al. [6] studied bidirectional ratcheted communication and proposed a protocol. However, their protocol does not offer 0-RTT and requires synchronized roles. At EuroS&P 2017, Cohn-Gordon et al. [5] formally studied Signal.

0-RTT communication with forward secrecy was achieved using puncturable encryption by Günther et al. at EUROCRYPT 2017 [9]. Later on, at EUROCRYPT 2018, Derler et al. made it reasonable practical by using Bloom filters [7].

At CRYPTO 2017, Bellare et al. [2] gave a secure ratcheting key exchange protocol. Their protocol is unidirectional and does not allow receiver exposure.

At CRYPTO 2018, Poettering and Rösler (PR) [15] studied bidirectional asynchronous ratcheted key agreement and presented a protocol which is secure in the random oracle model. Their solution further relies on hierarchical identity-based encryption (HIBE) but offers stronger security than required for practical usage, leaving ample room for improving the protocol. At the same conference, Jaeger and Stepanovs (JS) [10] had similar results but focused on secure communication rather than key agreement. They proposed another protocol relying on HIBE. In both results, HIBE is used to construct encryption/signature schemes with key-update security. This is a rather new notion allowing forward secrecy but is expensive to achieve. In both cases, it was claimed that the depth of HIBE is really small. However, when participants are disconnected and continue sending several messages, the depth increases rapidly. Consequently, HIBE needs unbounded depth.

At EUROCRYPT 2019, Jost, Maurer, and Mularczyk (JMM) [11] designed another ratcheting protocol which has "*near-optimal*" security, and does not use HIBE. Nevertheless, it still has a huge complexity: When messages alternate well (i.e., no participant sends two messages without receiving one in between), processing n messages requires $\mathcal{O}(n)$ operations in total. However, when messages accumulate before alternating (for instance, because the participants are disconnected by the network), the complexity becomes $\mathcal{O}(n^2)$. This is also the case for PR [15] and JS [10].[1] One advantage of the JMM protocol [11] comes with the resilience with random coin leakage as discussed below.

At EUROCRYPT 2019, Alwen, Coretti, and Dodis (ACD) [1] designed two other ratcheting protocols aiming at *instant decryption*, i.e. the ability to decrypt even though some previous messages have not been received yet. This is closer to real-life protocols but this comes with a potential threat: keys to decrypt un-delivered messages are stored until the messages are delivered. Hence, the adversary could choose to hold messages and decrypt them with future state exposure. This prevents forward secrecy. Furthermore, unless the direction of communication changes (or more precisely, if the *epoch* increases), their protocols are not really ratcheting as no random coins are used to update the state. This weakens post-compromise security as well. In Table 1, we call this weaker security "*id-optimal*" (not to say "insecure" in the model we are interested in) because it is the best we can obtain with immediate decryption. The lighter of the two protocols is not competing in the same category because it mostly uses symmetric cryptography. It is more efficient but with lower security. Namely, corrupting the state of a participant A implies impersonating B to A, and also decrypting the messages that A sends. Other protocols do not have this weakness. The second ACD protocol [1] (in the full version) uses asymmetric cryptography.

Some authors address the corruption of random coins in different ways. Bellare et al. [2] and JMM [11] allow leaking the random coins just *after* use. JS [10] allow leaking it just *before* usage only. ACD [1] allow adversarially *chosen* random coins. In most of the protocols, revealing (or choosing) the random coins imply revealing some part of the new state which allows decrypting incoming messages. It is comparable to state exposure. JMM [11] offers better security as revealing the random coins reveals the new state (and allows to decrypt) only when the previous state was already known.

---

[1] For JS, this is only visible in the corrected version of the paper on eprint [10]. Our complexity analysis is based on how those protocols have been implemented (https://github.com/qantik/ratcheted). It was presented at the WSM 2019 workshop.

Table 1. Comparison of protocols: complexity for exchanging n messages in alternating or accumulating mode, with timing (in seconds) for $n = 900$ of comparable implementations and asymptotic; and types of coin-leakage security ($\Rightarrow$ state exposure means coins leakage implies a state exposure).

	Security	Complexity		Coins leakage resilience	Model
		Alternating	Accumulating		
Poettering-Rösler [15]	Optimal	86.3, $\mathcal{O}(n)$	5897, $\mathcal{O}(n^2)$	No	ROM
Jaeger-Stepanovs [10]	Optimal	58.1, $\mathcal{O}(n)$	9087, $\mathcal{O}(n^2)$	Pre-send leakage, $\Rightarrow$ state exposure	ROM
Jost-Maurer-Mularczyk [11]	Near-optimal	2.08, $\mathcal{O}(n)$	11.4, $\mathcal{O}(n^2)$	Post-send leakage	ROM
BARK [this paper]	Sub-optimal	1.46, $\mathcal{O}(n)$	1.09, $\mathcal{O}(n)$	No	Plain
Alwen-Coretti-Dodis [1]	Id-optimal	1.18, $\mathcal{O}(n)$	0.92, $\mathcal{O}(n)$	Chosen coins, $\Rightarrow$ state exposure	Plain

Our Contributions. We give a definition for a bidirectional asynchronous key agreement (BARK) along with security properties. We start setting the stage with some definitions (such as *matching status*) then identify all cases leading to trivial attacks. We split them into *direct* and *indirect leakages*. Then, we define security with a KIND game (privacy). We also consider the resistance to forgery (impersonation) and the resistance to attacks which would heal after active attacks (RECOVER security). We use these two notions as building blocks to prove KIND-security. We finally construct a secure protocol. Our design choices are detailed below and compared to other papers.

1. **Simplicity.** Contrary to previous work, we define KIND security in a very comprehensive way by bringing all notions under the umbrella of a *cleanness* predicate which identifies and captures all trivial ways of attacking.
2. **Strong security.** In the same line as previous works, the adversary in our model can see the entire communication between participants and control the delivery. Of course, he can replace messages with anything. Scheduling communications is under the control of the adversary. This means that the time when a participant sends or receives messages is decided by the adversary. Moreover, the adversary is capable of corrupting participants by making exposures of their internal data. We separate two types of exposures: the exposure of the state (that is kept in internal machinery of a participant) and the exposure of the key (which is produced by the key agreement and given to an external protocol). This is because states are (normally) kept secure in our protocol while the generated key is transferred to other applications which may leak for different reasons. We do not consider exposure of the random coins.
3. **Slightly sub-optimal security.** Using the result from exposure allows the adversary to be active, e.g. by impersonating the exposed participant. However, the adversary is not allowed to use exposures to make a *trivial* attack. Identifying such trivial attacks is not easy. As a design goal, we adopt not to forbid more than what the intuitive notion of ratcheting captures. We do forbid a bit more than PR [15] and JS [10] which are considered of having *optimal*

security and than JMM [11] (which has *near-optimal* security)[2], though, allowing lighter building blocks. Namely, we need no key-update primitives and have linear-time complexity in terms of the number of exchanged messages, even when the network is occasionally down. **This translates to an important speedup factor**, as shown on Table 1. We argue that this is a reasonable choice enabling ratchet security as we define it: *unless trivial leakage, a message is private as long as it is acknowledged for reception in a subsequent message from the receiver.*

4. **Sequence integrity.** We believe that duplex communication is reliably enforced by a lower level protocol. This is assumed to solve non-malicious packet loss e.g. by resend requests and also to reconstruct the correct sequence order. What we only have to care of is when an adversary prevents the delivery of a message consistently. We make the choice to make the transmission of the next messages impossible under such an attack. Contrarily, ACD [1] advocates for immediate decryption, even though one message is missing. This lowers the security and we chose not to have it.

In the BARK protocol, the correctness implies that both participants generate the same keys. We define the stages *matching status, direct leakage, indirect leakage*. We aim to separate trivial attacks and trivial forgeries from non-trivial cases with our definitions. Direct and indirect leakages define when the adversary can trivially deduce the key generated due to the exposure of a participant who can either be the same participant (direct) or their counterpart (indirect).

We construct a secure BARK protocol. We build our constructions on top of a public-key cryptosystem and a signature scheme and achieve strong security, without key-update primitives or random oracles. We further show that a weakly secure unidirectional BARK implies public-key encryption.

Notations. We have two characters: Alice (A) and Bob (B). When P designates a participant, $\overline{\mathsf{P}}$ refers to P's counterpart. We use the roles send and rec for sender and receiver respectively. We define $\overline{\mathsf{send}} = \mathsf{rec}$ and $\overline{\mathsf{rec}} = \mathsf{send}$. When participants A and B have exclusive roles (like in unidirectional cases), we call them *sender* S and *receiver* R. PPT stands for *probabilistic polynomially bounded.* Negligible in terms of λ means in $\cap_{c>0}\mathcal{O}(\lambda^{-c})$ as $\lambda \to +\infty$.

Structure of the Paper. In Sect. 2, we define our BARK protocol along with correctness definition and KIND security. Section 3 proves that a simple unidirectional scheme implies public-key cryptography. In Sect. 4 we define the security notions unforgeability and unrecoverability. In Sect. 5, we give our BARK construction. Due to space limitation, some material was moved to the full version of this paper [8], including the definition of underlying primitives and the proofs of our results.

---

[2] Those terms are more formally explained on p. 12.

2 Bidirectional Asynchronous Ratcheted Communication

2.1 BARK Definition and Correctness

Definition 1 (BARK)**.** *A* bidirectional asynchronous ratcheted key agreement (BARK) *consists of the following PPT algorithms:*

- $\mathsf{Setup}(1^\lambda) \xrightarrow{\$} \mathsf{pp}$*: This defines the common public parameters* pp*.*
- $\mathsf{Gen}(1^\lambda, \mathsf{pp}) \xrightarrow{\$} (\mathsf{sk}, \mathsf{pk})$*: This generates the secret key* sk *and the public key* pk *of a participant.*
- $\mathsf{Init}(1^\lambda, \mathsf{pp}, \mathsf{sk}_\mathsf{P}, \mathsf{pk}_{\overline{\mathsf{P}}}, \mathsf{P}) \rightarrow \mathsf{st}_\mathsf{P}$*: This sets up the initial state* st_P *of* P *given his secret key and the public key of his counterpart.*
- $\mathsf{Send}(\mathsf{st}_\mathsf{P}) \xrightarrow{\$} (\mathsf{st}'_\mathsf{P}, \mathsf{upd}, \mathsf{k})$*: The algorithm inputs a current state* st_P *for* $\mathsf{P} \in \{\mathsf{A}, \mathsf{B}\}$*. It outputs a tuple* $(\mathsf{st}'_\mathsf{P}, \mathsf{upd}, \mathsf{k})$ *with an updated state* st'_P*, a message* upd*, and a key* k*.*
- $\mathsf{Receive}(\mathsf{st}_\mathsf{P}, \mathsf{upd}) \rightarrow (\mathsf{acc}, \mathsf{st}'_\mathsf{P}, \mathsf{k})$*: The algorithm inputs* $(\mathsf{st}_\mathsf{P}, \mathsf{upd})$ *where* $\mathsf{P} \in \{\mathsf{A}, \mathsf{B}\}$*. It outputs a triple consisting of a flag* $\mathsf{acc} \in \{\mathsf{true}, \mathsf{false}\}$ *to indicate an accept or reject of* upd *information, an updated state* st'_P*, and a key* k *i.e.* $(\mathsf{acc}, \mathsf{st}'_\mathsf{P}, \mathsf{k})$*.*

For convenience, we define the following initialization procedure for all games. It returns the initial states as well as some publicly available information z*.*

```
Initall($1^\lambda$, pp):
  $\mathsf{Gen}(1^\lambda, \mathsf{pp}) \rightarrow (\mathsf{sk}_\mathsf{A}, \mathsf{pk}_\mathsf{A})$
  $\mathsf{Gen}(1^\lambda, \mathsf{pp}) \rightarrow (\mathsf{sk}_\mathsf{B}, \mathsf{pk}_\mathsf{B})$
  $\mathsf{st}_\mathsf{A} \leftarrow \mathsf{Init}(1^\lambda, \mathsf{pp}, \mathsf{sk}_\mathsf{A}, \mathsf{pk}_\mathsf{B}, \mathsf{A})$
  $\mathsf{st}_\mathsf{B} \leftarrow \mathsf{Init}(1^\lambda, \mathsf{pp}, \mathsf{sk}_\mathsf{B}, \mathsf{pk}_\mathsf{A}, \mathsf{B})$
  $z \leftarrow (\mathsf{pp}, \mathsf{pk}_\mathsf{A}, \mathsf{pk}_\mathsf{B})$
  return $(\mathsf{st}_\mathsf{A}, \mathsf{st}_\mathsf{B}, z)$
```

Initialization is *splittable* in the sense that private keys can be generated by their holders with no need to rely on an authority (except maybe for authentication of pk_A and pk_B). Other protocols from the literature assume a trusted initialization.

We consider bidirectional asynchronous communications. We can see, in Fig. 1, Alice and Bob running some sequences of Send and $\mathsf{Receive}$ operations without any prior agreement. Their *time scale* is different. This means that Alice and Bob run algorithms in an asynchronous way. We consider a notion of *time* relative to a participant P. Formally, the time t for P is the number of elementary steps that P executed since the beginning of the game. We assume no common clock. However, events occur in a *game* and we may have to compare the time of two different participants by reference to the scheduling of the game. E.g., we could say that time t_A for A happens *before* time t_B for B. Normally, scheduling is under the control of the adversary except in the $\mathsf{CORRECT}$ game in which there is no adversary. There, we define the scheduling by a sequence of actions. Reading the sequence tells who executes a new step of the protocol.

The protocol also uses random roles. Alice and Bob can both send and receive messages. They take their role (sender or receiver) in a sequence, but the sequences of roles of Alice and Bob are not necessarily synchronized. Sending/receiving is refined by the $\mathsf{RATCH}(\mathsf{P}, \mathsf{role}, [\mathsf{upd}])$ call in Fig. 2.

Correctness. We say that a ratcheted communication protocol functions correctly if the receiver accepts the update information upd and generates the same key as its counterpart. Correctness implies that the received keys for participant P have been generated in the same order as sent keys of participant $\overline{\mathsf{P}}$. We formally define the $\mathsf{CORRECT}$ game in Fig. 2. We define variables. $\mathsf{received}_{\mathsf{key}}^{\mathsf{P}}$ (respectively $\mathsf{sent}_{\mathsf{key}}^{\mathsf{P}}$) keeps a list of secret keys that are generated by P when running $\mathsf{Receive}$ (respectively, Send). Similarly, $\mathsf{received}_{\mathsf{msg}}^{\mathsf{P}}$ (respectively $\mathsf{sent}_{\mathsf{msg}}^{\mathsf{P}}$) keeps a list of upd information that are received (respectively sent) by P and accepted by $\mathsf{Receive}$. The $\mathsf{received}$ sequences only keep values for which $\mathsf{acc} = \mathsf{true}$.

Each variable v such as $\mathsf{received}_{\mathsf{msg}}^{\mathsf{P}}$, k_{P}, or st_{P} is relative to a participant P. We denote by $v(\mathsf{t})$ the value of v at time t for P. For instance, $\mathsf{received}_{\mathsf{msg}}^{A}(\mathsf{t})$ is the sequence of upd which were received and accepted by A at time t for A.

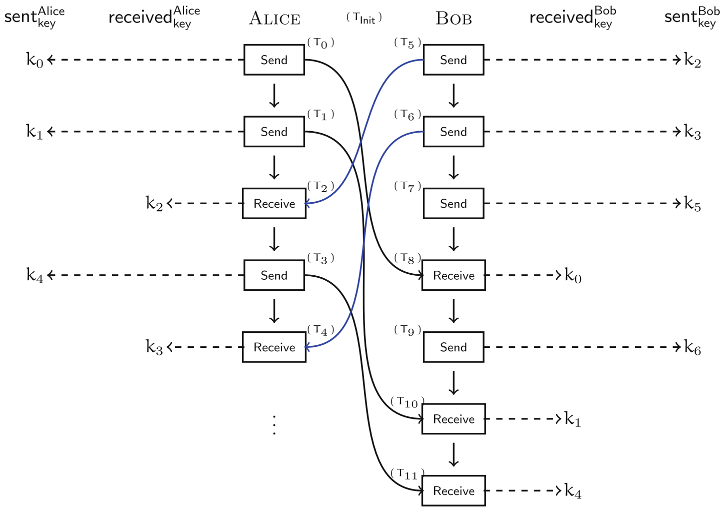


Fig. 1. The message exchange between Alice and Bob.

We initialize the two participants in the $\mathsf{CORRECT}$ game in Fig. 2. The scheduling is defined by a sequence sched of tuples of form either $(\mathsf{P}, \mathsf{send})$ (saying that P must send) or $(\mathsf{P}, \mathsf{rec})$ (saying that P must receive). In this game, communication between the participants uses a waiting queue for messages in each direction. Each participant has a queue of incoming messages and is pulling them in the order they have been pushed in. Sent messages from P are buffered in the queue of $\overline{\mathsf{P}}$.

```
Oracle $\mathsf{RATCH}(\mathsf{P}, \mathsf{rec}, \mathsf{upd})$
$(\mathsf{acc}, \mathsf{st}'_\mathsf{P}, \mathsf{k}) \leftarrow \mathsf{Receive}(\mathsf{st}_\mathsf{P}, \mathsf{upd})$
if acc then
    $\mathsf{upd}_\mathsf{P} \leftarrow \mathsf{upd}$
    $\mathsf{k}_\mathsf{P} \leftarrow \mathsf{k}$
    $\mathsf{st}_\mathsf{P} \leftarrow \mathsf{st}'_\mathsf{P}$
    append $\mathsf{k}_\mathsf{P}$ to $\mathsf{received}^\mathsf{P}_\mathsf{key}$
    append $\mathsf{upd}_\mathsf{P}$ to $\mathsf{received}^\mathsf{P}_\mathsf{msg}$
end if
return acc

Oracle $\mathsf{RATCH}(\mathsf{P}, \mathsf{send})$
$(\mathsf{st}'_\mathsf{P}, \mathsf{upd}_\mathsf{P}, \mathsf{k}_\mathsf{P}) \leftarrow \mathsf{Send}(\mathsf{st}_\mathsf{P})$
$\mathsf{st}_\mathsf{P} \leftarrow \mathsf{st}'_\mathsf{P}$
append $\mathsf{k}_\mathsf{P}$ to $\mathsf{sent}^\mathsf{P}_\mathsf{key}$
append $\mathsf{upd}_\mathsf{P}$ to $\mathsf{sent}^\mathsf{P}_\mathsf{msg}$
return $\mathsf{upd}_\mathsf{P}$
```

```
Game $\mathsf{CORRECT}(1^\lambda, \mathsf{sched})$
set all $\mathsf{sent}^*_*$ and $\mathsf{received}^*_*$ variables to $\emptyset$
$\mathsf{Setup}(1^\lambda) \xrightarrow{\$} \mathsf{pp}$
$\mathsf{Initall}(1^\lambda, \mathsf{pp}) \xrightarrow{\$} (\mathsf{st}_A, \mathsf{st}_B, z)$
initialize two FIFO lists $\mathsf{incoming}_A$ and $\mathsf{incoming}_B$ to empty
$i \leftarrow 1$
while $\mathsf{sched}_i$ exists do
    $(\mathsf{P}, \mathsf{role}) \leftarrow \mathsf{sched}_i$
    if $\mathsf{role} = \mathsf{rec}$ then
        if $\mathsf{incoming}_\mathsf{P}$ is empty then return 0
        pull upd from $\mathsf{incoming}_\mathsf{P}$
        $\mathsf{acc} \leftarrow \mathsf{RATCH}(\mathsf{P}, \mathsf{rec}, \mathsf{upd})$
        if $\mathsf{acc} = \mathsf{false}$ then return 1
    else
        $\mathsf{upd} \leftarrow \mathsf{RATCH}(\mathsf{P}, \mathsf{send})$
        push upd to $\mathsf{incoming}_{\overline{\mathsf{P}}}$
    end if
    if $\mathsf{received}^A_\mathsf{key}$ not prefix of $\mathsf{sent}^B_\mathsf{key}$ then return 1
    if $\mathsf{received}^B_\mathsf{key}$ not prefix of $\mathsf{sent}^A_\mathsf{key}$ then return 1
    $i \leftarrow i + 1$
end while
return 0
```

Fig. 2. The CORRECT game.

Definition 2 (Correctness of BARK). *We say that* BARK *is* correct *if for all sequence* sched*, the* CORRECT *game of Fig. 2 never returns 1. Namely, for each* P*,* $\mathsf{received}^\mathsf{P}_\mathsf{key}$ *is always prefix of* $\mathsf{sent}^{\overline{\mathsf{P}}}_\mathsf{key}$ [3] *and each* $\mathsf{RATCH}(., \mathsf{rec}, .)$ *call accepts.*

Security. We model our security notion with an active adversary who can have access to some of the states of Alice or Bob along with access to their secret keys enabling them to act both as a sender and as a receiver. For simplicity, we have only Alice and Bob as participants. (Models with more participants would be asymptotically equivalent.) We focus on three main security notions which are *key indistinguishability* (denoted as KIND) under the compromise of states or keys, *unforgeability* of upd information (FORGE) by the adversary which will be accepted, and *recovery from impersonation* (RECOVER) which will make the two participants restore secure communication without noticing a (trivial) impersonation resulting from a state exposure. A challenge in these notions is to eliminate the trivial attacks. FORGE and RECOVER security will be useful to prove KIND security.

2.2 KIND Security

The adversary can access four oracles called RATCH, EXP_st, $\mathsf{EXP}_\mathsf{key}$, and TEST.

---

[3] By saying that $\mathsf{received}^\mathsf{P}_\mathsf{key}$ is prefix of $\mathsf{sent}^{\overline{\mathsf{P}}}_\mathsf{key}$, we mean that when n is the number of keys generated by P running Receive, then these keys are the first n keys generated by $\overline{\mathsf{P}}$ running Send.

RATCH. This is essentially the message exchange procedure. It is defined in Fig. 2. The adversary can call it with three inputs, a participant P, where $\mathsf{P} \in \{\mathsf{A}, \mathsf{B}\}$; a role of P; and an upd information if the role is rec. The adversary gets upd (for $\mathsf{role} = \mathsf{send}$) or acc (for $\mathsf{role} = \mathsf{rec}$) in return.

$\mathsf{EXP}_{\mathsf{st}}$. The adversary can expose the state of Alice or Bob. It inputs $\mathsf{P} \in \{\mathsf{A}, \mathsf{B}\}$ to the $\mathsf{EXP}_{\mathsf{st}}$ oracle and it receives the full state st_{P} of P.

$\mathsf{EXP}_{\mathsf{key}}$. The adversary can expose the generated key by calling this oracle. Upon inputting P, it gets the last key k_{P} generated by P. If no key was generated, $\perp$ is returned.

TEST. This oracle can be called only once to receive a challenge key which is generated either uniformly at random (if the challenge bit is $\mathsf{b} = 0$) or given as the last generated key of a participant P specified as input (if the challenge bit is $\mathsf{b} = 1$). The oracle cannot be queried if no key was generated yet.

We specifically separate $\mathsf{EXP}_{\mathsf{key}}$ from $\mathsf{EXP}_{\mathsf{st}}$ because the key k generated by BARK will be used by an external process which may leak the key. Thus, $\mathsf{EXP}_{\mathsf{key}}$ can be more frequent than $\mathsf{EXP}_{\mathsf{st}}$, however it harms security less.

To define security, we avoid trivial attacks. Capturing the trivial cases in a broad sense requires a new set of definitions. All of them are intuitive.

Intuitively, P is in a matching status at a given time if his state is not dependent on an active attack (i.e. could result from a $\mathsf{CORRECT}$ game).

Definition 3 (Matching status). *We say that* P *is in a* matching status *at time* t *for* P *if 1. at any moment of the game before time* t *for* P*,* $\mathsf{received}_{\mathsf{msg}}^{\mathsf{P}}$ *is a prefix of* $\mathsf{sent}_{\mathsf{msg}}^{\overline{\mathsf{P}}}$*—this defines the time* $\overline{\mathsf{t}}$ *for* $\overline{\mathsf{P}}$ *when* $\overline{\mathsf{P}}$ *sent the last message in* $\mathsf{received}_{\mathsf{msg}}^{\mathsf{P}}(\mathsf{t})$*; 2. at any moment of the game before time* $\overline{\mathsf{t}}$ *for* $\overline{\mathsf{P}}$*,* $\mathsf{received}_{\mathsf{msg}}^{\overline{\mathsf{P}}}$ *is a prefix of* $\mathsf{sent}_{\mathsf{msg}}^{\mathsf{P}}$*. We further say that time* t *for* P originates *from time* $\overline{\mathsf{t}}$ *for* $\overline{\mathsf{P}}$*.*

The first condition clearly states that each of the received (and accepted) upd message was sent before by the counterpart of P, in the same order, without any loss in between. The second condition similarly verifies that those messages from $\overline{\mathsf{P}}$ only depend on information coming from P. In Fig. 1, Bob is in a matching status with Alice because he receives the upd information in the exact order as they have sent by Alice (i.e. Bob generates k_2 after k_1 and k_4 after k_2 same as it has sent by Alice). In general, as long as no adversary switches the order of messages or creates fake messages successfully for either party, the participants are always in a matching status.

The key exchange literature often defines a notion of partnering which is simpler. Here, asynchronous random roles make it more complicated.

Here is an easy property of the notion of matching status.

Lemma 4. *If* P *is in a matching status at time* t*, then* P *is also in a matching status at any time* $\mathsf{t}_0 \leqslant \mathsf{t}$*. Similarly, if* P *is in a matching status at time* t *and* t *for* P *originates from* $\overline{\mathsf{t}}$ *for* $\overline{\mathsf{P}}$*, then* $\overline{\mathsf{P}}$ *is in a matching status at time* $\overline{\mathsf{t}}$*.*

Definition 5 (Forgery). *Given a participant* P *in a game, we say that* $\mathsf{upd} \in \mathsf{received}_{\mathsf{msg}}^{\mathsf{P}}$ *is a forgery if at the moment of the game just before* P *received* upd*,* P *was in a matching status, but no longer after receiving* upd*.*

In a matching status, any upd received by P must correspond to an upd sent by $\overline{\mathsf{P}}$ and the sequences must match. This implies the following notion.

Definition 6 (Corresponding RATCH calls). *Let* P *be a participant. We consider the* $\mathsf{RATCH}(\mathsf{P}, \mathsf{rec}, .)$ *calls by* P *returning* true*. We say that the* i^{th} *receiving call* corresponds *to the* j^{th} $\mathsf{RATCH}(\overline{\mathsf{P}}, \mathsf{send})$ *call if* $\mathsf{i} = \mathsf{j}$ *and* P *is in matching status at the time of this* i^{th} *accepting* $\mathsf{RATCH}(\mathsf{P}, \mathsf{rec}, .)$ *call.*

Lemma 7. *In a correct* BARK *protocol, two corresponding* $\mathsf{RATCH}(\mathsf{P}, \mathsf{rec}, \mathsf{upd})$ *and* $\mathsf{RATCH}(\overline{\mathsf{P}}, \mathsf{send})$ *calls generate the same key* $\mathsf{k}_{\mathsf{P}} = \mathsf{k}_{\overline{\mathsf{P}}}$*.*

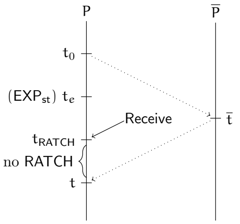
Definition 8 (Ratcheting period of P). *A maximal time interval during which there is no* $\mathsf{RATCH}(\mathsf{P}, \mathsf{send})$ *call is called a* ratcheting period *of* P*.*

In Fig. 1, the intervals $\mathsf{T}_1 - \mathsf{T}_3$ and $\mathsf{T}_5 - \mathsf{T}_6$ are ratcheting periods.

We now define when the adversary can trivially obtain a key generated by P due to an exposure. We distinguish the case when the exposure was done on P (direct leakage) and on $\overline{\mathsf{P}}$ (indirect leakage).

Definition 9 (Direct leakage). *Let* t *be a time and* P *be a participant. We say that* $\mathsf{k}_{\mathsf{P}}(\mathsf{t})$ *has a* direct leakage *if one of the following conditions is satisfied:*

- *There is an* $\mathsf{EXP}_{\mathsf{key}}(\mathsf{P})$ *at a time* t_{e} *such that the last* RATCH *call which is executed by* P *before time* t *and the last* RATCH *call which is executed by* P *before time* t_{e} *are the same.*
- P *is in a matching status and there exists* $\mathsf{t}_0 \leqslant \mathsf{t}_{\mathsf{e}} \leqslant \mathsf{t}_{\mathsf{RATCH}} \leqslant \mathsf{t}$ *and* $\overline{\mathsf{t}}$ *such that time* t *originates from time* $\overline{\mathsf{t}}$*; time* $\overline{\mathsf{t}}$ *originates from time* t_0*; there is one* $\mathsf{EXP}_{\mathsf{st}}(\mathsf{P})$ *at time* t_{e}*; there is one* $\mathsf{RATCH}(\mathsf{P}, \mathsf{rec}, .)$ *at time* $\mathsf{t}_{\mathsf{RATCH}}$*; and there is no* $\mathsf{RATCH}(\mathsf{P}, ., .)$ *between time* $\mathsf{t}_{\mathsf{RATCH}}$ *and time* t*.*



In the first case, it is clear that $\mathsf{EXP}_{\mathsf{key}}(\mathsf{P})$ gives $\mathsf{k}_{\mathsf{P}}(\mathsf{t}_{\mathsf{e}}) = \mathsf{k}_{\mathsf{P}}(\mathsf{t})$. In the second case (in the figure[4]), the state which leaks from $\mathsf{EXP}_{\mathsf{st}}(\mathsf{P})$ at time t_{e} allows to simulate all deterministic Receive (by skipping all Send) and to compute the key $\mathsf{k}_{\mathsf{P}}(\mathsf{t}_{\mathsf{RATCH}}) = \mathsf{k}_{\mathsf{P}}(\mathsf{t})$. The reason why we can allow the adversary to skip all Send is that they make messages which are supposed to be delivered to $\overline{\mathsf{P}}$ after time $\overline{\mathsf{t}}$, so they have no impact on $\mathsf{k}_{\mathsf{P}}(\mathsf{t})$.

Consider Fig. 1. Suppose t is in between time T_3 and T_4. According to our definition $\mathsf{P} = \mathsf{A}$ and the last RATCH call is at time T_3. It is a Send, thus the

---

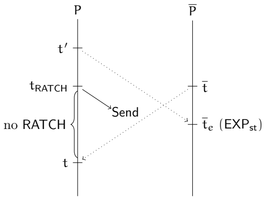
[4] Origin of dotted arrows indicate when a time originates from.

second case cannot apply. The next RATCH call is at time T_4. In this case, $\mathsf{k}_\mathsf{A}(\mathsf{t})$ has a direct leakage if there is a key exposure of Alice between T_3 and T_4.

Suppose now that $\mathsf{T}_8 < \mathsf{t} < \mathsf{T}_9$. We have $\mathsf{P} = \mathsf{B}$, the last RATCH call is a $\mathsf{Receive}$, it is at time $\mathsf{t}_\mathsf{RATCH} = \mathsf{T}_8$, and t originates from time $\bar{\mathsf{t}} = \mathsf{T}_0$ which itself originates from the origin time $\mathsf{t}_0 = \mathsf{T}_\mathsf{Init}$ for B. We say that t has a direct leakage if there is a key exposure between $\mathsf{T}_8 - \mathsf{T}_9$ or a state exposure of Bob before time T_8. Indeed, with this last state exposure, the adversary can ignore all Send and simulate all $\mathsf{Receive}$ to derive k_0.

Definition 10 (Indirect leakage). *We consider a time* t *and a participant* P. *Let* $\mathsf{t}_\mathsf{RATCH}$ *be the time of the last successful* RATCH *call and* role *be its input role. (We have* $\mathsf{k}_\mathsf{P}(\mathsf{t}_\mathsf{RATCH}) = \mathsf{k}_\mathsf{P}(\mathsf{t})$.*) We say that* $\mathsf{k}_\mathsf{P}(\mathsf{t})$ *has an* indirect leakage *if* P *is in matching status at time* t *and one of the following conditions is satisfied*

- *There exists a* $\mathsf{RATCH}(\overline{\mathsf{P}}, \overline{\mathsf{role}}, .)$ *corresponding to that* $\mathsf{RATCH}(\mathsf{P}, \mathsf{role}, .)$ *and making a* $\mathsf{k}_{\overline{\mathsf{P}}}$ *which has a direct leakage for* $\overline{\mathsf{P}}$.
- *There exists* $\mathsf{t}' \leqslant \mathsf{t}_\mathsf{RATCH} \leqslant \mathsf{t}$ *and* $\bar{\mathsf{t}} \leqslant \bar{\mathsf{t}}_\mathsf{e}$ *such that* $\overline{\mathsf{P}}$ *is in a matching status at time* $\bar{\mathsf{t}}_\mathsf{e}$, t *originates from* $\bar{\mathsf{t}}$, $\bar{\mathsf{t}}_\mathsf{e}$ *originates from* t', *there is one* $\mathsf{EXP}_\mathsf{st}(\overline{\mathsf{P}})$ *at time* $\bar{\mathsf{t}}_\mathsf{e}$, *and* $\mathsf{role} = \mathsf{send}$.



In the first case, $\mathsf{k}_\mathsf{P}(\mathsf{t}) = \mathsf{k}_\mathsf{P}(\mathsf{t}_\mathsf{RATCH})$ is also computed by $\overline{\mathsf{P}}$ and leaks from there. The second case (in the figure) is more complicated: it corresponds to an adversary who can get the internal state of $\overline{\mathsf{P}}$ by $\mathsf{EXP}_\mathsf{st}(\overline{\mathsf{P}})$ then simulate all $\mathsf{Receive}$ with messages from P until the one sent at time $\mathsf{t}_\mathsf{RATCH}$, ignoring all Send by $\overline{\mathsf{P}}$, to recover $\mathsf{k}_\mathsf{P}(\mathsf{t})$.

For example, let t be a time between T_1 and T_2 in Fig. 1. We take $\mathsf{P} = \mathsf{A}$. The last RATCH call is at time $\mathsf{t}_\mathsf{RATCH} = \mathsf{T}_1$, it is a Send and corresponds to a $\mathsf{Receive}$ at time T_{10}, but t originates from time $\bar{\mathsf{t}} = \mathsf{T}_\mathsf{Init}$. We say that t has an indirect leakage for A if there exists a direct leakage for $\overline{\mathsf{P}} = \mathsf{B}$ at a time between T_{10} and T_{11} (first condition) or there exists a $\mathsf{EXP}_\mathsf{st}(\mathsf{B})$ call at a time $\bar{\mathsf{t}}_\mathsf{e}$ (after time $\bar{\mathsf{t}} = \mathsf{T}_\mathsf{Init}$), originating from a time t' before time T_1, so $\bar{\mathsf{t}}_\mathsf{e} < \mathsf{T}_{10}$ (second condition). In the latter case, the adversary can simulate $\mathsf{Receive}$ with the updates sent at time T_0 and T_1 to derive the key k_1.

Exposing the state of a participant gives certain advantages to the attacker and make trivial attacks possible. In our security game, we avoid those attack scenarios. In the following lemma, we show that direct and indirect leakage capture the times when the adversary can trivially win. The proof is straightforward.

Lemma 11 (Trivial attacks). *Assume that* BARK *is correct. For any* t *and* P, *if* $\mathsf{k}_\mathsf{P}(\mathsf{t})$ *has a direct or indirect leakage, the adversary can compute* $\mathsf{k}_\mathsf{P}(\mathsf{t})$.

So far, we mostly focused on matching status cases but there could be situations with forgeries. Some are unavoidable. We call them *trivial* forgeries.

Definition 12 (Trivial forgery). *Let* upd *be a forgery received by* P*. At the time* t *just before the* $\mathsf{RATCH}(\mathsf{P}, \mathsf{rec}, \mathsf{upd})$ *call,* P *was in a matching status. We assume that time* t *for* P *originates from time* $\overline{\mathsf{t}}$ *for* $\overline{\mathsf{P}}$*. If there is an* $\mathsf{EXP}_{\mathsf{st}}(\overline{\mathsf{P}})$ *call during the ratcheting period of* $\overline{\mathsf{P}}$ *starting at time* $\overline{\mathsf{t}}$*, we say that* upd *is a* trivial forgery*.*

We define the KIND security game in Fig. 3. Essentially, the adversary plays with all oracles. At some point, he does one $\mathsf{TEST}(\mathsf{P})$ call which returns either the same result as $\mathsf{EXP}_{\mathsf{key}}(\mathsf{P})$ (case $\mathsf{b} = 1$) or some random value (case $\mathsf{b} = 0$). The goal of the adversary is to guess b. The TEST call can be done only once and it defines the participant $\mathsf{P}_{\mathsf{test}} = \mathsf{P}$ and the time $\mathsf{t}_{\mathsf{test}}$ at which this call is made. It also defines $\mathsf{upd}_{\mathsf{test}}$, the last upd which was used (either sent or received) to carry $\mathsf{k}_{\mathsf{P}_{\mathsf{test}}}(\mathsf{t}_{\mathsf{test}})$ from the sender to the receiver. It is not allowed to make this call at the beginning, when P did not generate a key yet. It is not allowed to make a trivial attack as defined by a cleanness predicate $\mathsf{C}_{\mathsf{clean}}$ appearing on Step 6 in the KIND game in Fig. 3. Identifying the appropriate *cleanness predicate* $\mathsf{C}_{\mathsf{clean}}$ is not easy. It must clearly forbid trivial attacks but also allow efficient protocols. In what follows we use the following predicates:

- $\mathsf{C}_{\mathsf{leak}}$: $\mathsf{k}_{\mathsf{P}_{\mathsf{test}}}(\mathsf{t}_{\mathsf{test}})$ has no direct or indirect leakage.
- $\mathsf{C}^{\mathsf{P}}_{\mathsf{trivial\ forge}}$: P received no trivial forgery until P has seen $\mathsf{upd}_{\mathsf{test}}$.
  (This implies that $\mathsf{upd}_{\mathsf{test}}$ is not a trivial forgery. It also implies that if P never sees $\mathsf{upd}_{\mathsf{test}}$, then P received no trivial forgery at all.)
- $\mathsf{C}^{\mathsf{P}}_{\mathsf{forge}}$: P received no forgery until P has seen $\mathsf{upd}_{\mathsf{test}}$.
- $\mathsf{C}_{\mathsf{ratchet}}$: $\mathsf{upd}_{\mathsf{test}}$ was sent by a participant P, then received and accepted by $\overline{\mathsf{P}}$, then some $\mathsf{upd}_{\mathsf{ack}}$ was sent by $\overline{\mathsf{P}}$, then $\mathsf{upd}_{\mathsf{ack}}$ was received and accepted by P. (Here, P could be $\mathsf{P}_{\mathsf{test}}$ or his counterpart. This accounts for the receipt of $\mathsf{upd}_{\mathsf{test}}$ being acknowledged by $\overline{\mathsf{P}}$ through $\mathsf{upd}_{\mathsf{ack}}$.)
- $\mathsf{C}_{\mathsf{noEXP(R)}}$: there is no $\mathsf{EXP}_{\mathsf{st}}(\mathsf{R})$ and no $\mathsf{EXP}_{\mathsf{key}}(\mathsf{R})$ query. (R is the receiver.)

Lemma 11 says that the adopted cleanness predicate $\mathsf{C}_{\mathsf{clean}}$ must imply $\mathsf{C}_{\mathsf{leak}}$ in all considered games. Otherwise, no security is possible. It is however not sufficient as it only hardly trivial attacks with forgeries.

$\mathsf{C}_{\mathsf{ratchet}}$ targets that any acknowledged sent message is secure. Another way to say is that a key generated by one Send starting a round trip must be safe. This is the notion of healing by ratcheting. Intuitively, the security notion from $\mathsf{C}_{\mathsf{clean}} = \mathsf{C}_{\mathsf{leak}} \wedge \mathsf{C}_{\mathsf{ratchet}}$ is fair enough.

Bellare et al. [2] consider unidirectional BARK with $\mathsf{C}_{\mathsf{clean}} = \mathsf{C}_{\mathsf{leak}} \wedge \mathsf{C}^{\mathsf{P}_{\mathsf{test}}}_{\mathsf{trivial\ forge}} \wedge \mathsf{C}_{\mathsf{noEXP(R)}}$. Other papers like PR [15] and JS [10] implicitly use $\mathsf{C}_{\mathsf{clean}} = \mathsf{C}_{\mathsf{leak}} \wedge \mathsf{C}^{\mathsf{P}_{\mathsf{test}}}_{\mathsf{trivial\ forge}}$ as cleanness predicate. They show that this is sufficient to build secure protocols but it is probably not the minimal cleanness predicate. (It is nevertheless called "*optimal*".) JMM [11] excludes cases where $\overline{\mathsf{P}}_{\mathsf{test}}$ received a (trivial) forgery then had an $\mathsf{EXP}_{\mathsf{st}}(\overline{\mathsf{P}}_{\mathsf{test}})$ before receiving $\mathsf{upd}_{\mathsf{test}}$. Actually, they use a cleanness predicate ("*near-optimal*" security) which is somewhere between $\mathsf{C}_{\mathsf{leak}} \wedge \mathsf{C}^{\mathsf{P}_{\mathsf{test}}}_{\mathsf{trivial\ forge}}$ and $\mathsf{C}_{\mathsf{leak}} \wedge \mathsf{C}^{\mathsf{A}}_{\mathsf{trivial\ forge}} \wedge \mathsf{C}^{\mathsf{B}}_{\mathsf{trivial\ forge}}$: this cleanness implies the JMM cleanness which itself implies the PR/JS cleanness.

In our construction (“*sub-optimal*”), we use the predicate $\mathsf{C}_{\mathsf{clean}} = \mathsf{C}_{\mathsf{leak}} \wedge \mathsf{C}^{\mathsf{A}}_{\mathsf{forge}} \wedge \mathsf{C}^{\mathsf{B}}_{\mathsf{forge}}$. However, in Sect. 4.1, we define the FORGE security (unforgeability) which implies that $(\mathsf{C}_{\mathsf{leak}} \wedge \mathsf{C}^{\mathsf{A}}_{\mathsf{forge}} \wedge \mathsf{C}^{\mathsf{B}}_{\mathsf{forge}})$-KIND security and $(\mathsf{C}_{\mathsf{leak}} \wedge \mathsf{C}^{\mathsf{A}}_{\mathsf{trivial\ forge}} \wedge \mathsf{C}^{\mathsf{B}}_{\mathsf{trivial\ forge}})$-KIND security are equivalent. (See Theorem 16.) One drawback is that it forbids more than $(\mathsf{C}_{\mathsf{leak}} \wedge \mathsf{C}^{\mathsf{P}_{\mathsf{test}}}_{\mathsf{trivial\ forge}})$-KIND security. The advantage is that we can achieve security without key-update primitives. We will prove in Theorem 19 that this security is enough to achieve security with the predicate $\mathsf{C}_{\mathsf{clean}} = \mathsf{C}_{\mathsf{leak}} \wedge \mathsf{C}_{\mathsf{ratchet}}$, thanks to RECOVER-security which we define in Sect. 4.2. Thus, our cleanness notion is fair enough.

```
Game $\mathsf{KIND}^{\mathcal{A}}_{b,\mathsf{C}_{\mathsf{clean}}}(1^\lambda)$
  $\mathsf{Setup}(1^\lambda) \xrightarrow{\$} \mathsf{pp}$
  $\mathsf{InitAll}(1^\lambda, \mathsf{pp}) \xrightarrow{\$} (\mathsf{st}_A, \mathsf{st}_B, z)$
  set all $\mathsf{sent}^*_*$ and $\mathsf{received}^*_*$ variables to $\emptyset$
  set $t_{\mathsf{test}}$, $k_A$, $k_B$ to $\perp$
  $b' \leftarrow \mathcal{A}^{\mathsf{RATCH},\mathsf{EXP}_{\mathsf{st}},\mathsf{EXP}_{\mathsf{key}},\mathsf{TEST}}(z)$
  if $\neg\mathsf{C}_{\mathsf{clean}}$ then return $\perp$
  return $b'$

Oracle $\mathsf{EXP}_{\mathsf{st}}(P)$
  return $\mathsf{st}_P$
```

```
Oracle $\mathsf{TEST}(P)$
  if $t_{\mathsf{test}} \neq \perp$ then return $\perp$
  if $k_P = \perp$ then return $\perp$
  $t_{\mathsf{test}} \leftarrow \mathsf{time}$, $P_{\mathsf{test}} \leftarrow P$, $\mathsf{upd}_{\mathsf{test}} \leftarrow \mathsf{upd}_P$
  if $b = 1$ then
      return $k_P$
  else
      return random $\{0,1\}^{|k_P|}$
  end if

Oracle $\mathsf{EXP}_{\mathsf{key}}(P)$
  return $k_P$
```

Fig. 3. $\mathsf{C}_{\mathsf{clean}}$-KIND game. (Oracle RATCH is defined in Fig. 2.)

Definition 13 ($\mathsf{C}_{\mathsf{clean}}$-KINDsecurity). *Let* $\mathsf{C}_{\mathsf{clean}}$ *be a cleanness predicate. We consider the* $\mathsf{KIND}^{\mathcal{A}}_{b,\mathsf{C}_{\mathsf{clean}}}$ *game of Fig. 3. We say that the ratcheted key agreement* BARK *is* $\mathsf{C}_{\mathsf{clean}}$-KIND-*secure if for any PPT adversary, the advantage*

$$\mathsf{Adv}_{\mathcal{A}}(1^\lambda) = \left|\Pr\left[\mathsf{KIND}^{\mathcal{A}}_{0,\mathsf{C}_{\mathsf{clean}}}(1^\lambda) \to 1\right] - \Pr\left[\mathsf{KIND}^{\mathcal{A}}_{1,\mathsf{C}_{\mathsf{clean}}}(1^\lambda) \to 1\right]\right|$$

of $\mathcal{A}$ *in* $\mathsf{KIND}^{\mathcal{A}}_{b,\mathsf{C}_{\mathsf{clean}}}(1^\lambda)$ *security game is negligible.*

3 uniARK Implies KEM

We now prove that a weakly secure uniARK (a unidirectional asynchronous ratcheted key exchange—a straightforward variant of BARK in which messages can only be sent from a participant whom we call S and can only be received by another participant whom we call R) implies public key encryption. Namely, we can construct a key encapsulation mechanism (KEM) out of it. We recall the KEM definition in the full version [8]. We consider a uniARK which is KIND-secure for the following cleanness predicate:

$\mathsf{C}_{\mathsf{weak}}$: the adversary makes only three oracle calls which are, in order, $\mathsf{EXP}_{\mathsf{st}}(S)$, $\mathsf{RATCH}(S, \mathsf{send})$, and $\mathsf{TEST}(S)$.

(Note that R is never used.) $\mathsf{C}_{\mathsf{weak}}$ implies cleanness for all other considered predicates. Hence, it is more restrictive. Our result implies that it is unlikely to construct even such weakly secure uniARK from symmetric cryptography.

Theorem 14. *Given a* uniARK *protocol, we can construct a* KEM *with the following properties. The correctness of* uniARK *implies the correctness of* KEM*. The* $\mathsf{C}_{\mathsf{weak}}$*-*$\mathsf{KIND}$*-security of* uniARK *implies the* $\mathsf{IND\text{-}CPA}$ *security of* KEM*.*

Proof. Assuming a uniARK protocol, we construct a KEM as follows:

$\mathsf{KEM.Gen}(1^\lambda) \xrightarrow{\$} (\mathsf{sk},\mathsf{pk})$: run $\mathsf{uniARK.Setup}(1^\lambda) \xrightarrow{\$} \mathsf{pp}$, $\mathsf{uniARK.Initall}(1^\lambda,\mathsf{pp}) \xrightarrow{\$} (\mathsf{st}_S,\mathsf{st}_R,z)$, and set $\mathsf{pk} = \mathsf{st}_S$, $\mathsf{sk} = \mathsf{st}_R$.

$\mathsf{KEM.Enc}(\mathsf{pk}) \xrightarrow{\$} (\mathsf{k},\mathsf{ct})$: run $\mathsf{uniARK.Send}(\mathsf{pk}) \xrightarrow{\$} (.,\mathsf{upd},\mathsf{k})$ and set $\mathsf{ct} = \mathsf{upd}$.

$\mathsf{KEM.Dec}(\mathsf{sk},\mathsf{ct}) \to \mathsf{k}$: run $\mathsf{uniARK.Receive}(\mathsf{sk},\mathsf{upd}) \to (.,.,\mathsf{k})$.

The $\mathsf{IND\text{-}CPA}$ security game with adversary $\mathcal{A}$ works as in the left-hand side below. We transform $\mathcal{A}$ into a KIND adversary $\mathcal{B}$ in the right-hand side below.

```
Game IND-CPA:
    $\mathsf{KEM.Gen} \xrightarrow{\$} (\mathsf{sk},\mathsf{pk})$
    $\mathsf{KEM.Enc}(\mathsf{pk}) \xrightarrow{\$} (\mathsf{k},\mathsf{ct})$
    if $\mathsf{b} = 0$ then set $\mathsf{k}$ to random
    $\mathcal{A}(\mathsf{pk},\mathsf{ct},\mathsf{k}) \xrightarrow{\$} \mathsf{b}'$
    return $\mathsf{b}'$
```

```
Adversary $\mathcal{B}(z)$:
    call $\mathsf{EXP}_{\mathsf{st}}(S) \to \mathsf{pk}$
    call $\mathsf{RATCH}(S,\mathsf{send}) \to \mathsf{ct}$
    call $\mathsf{TEST}(S) \to \mathsf{k}$
    run $\mathcal{A}(\mathsf{pk},\mathsf{ct},\mathsf{k}) \to \mathsf{b}'$
    return $\mathsf{b}'$
```

We can check that $\mathsf{C}_{\mathsf{weak}}$ is satisfied. The KIND game with $\mathcal{B}$ simulates perfectly the $\mathsf{IND\text{-}CPA}$ game with $\mathcal{A}$. So, the KIND-security of uniARK implies the $\mathsf{IND\text{-}CPA}$ security of KEM. ☐

4 FORGE and RECOVER Security

4.1 Unforgeability

Another security aspect of the key agreement BARK is to have that no upd information is forgeable by any bounded adversary except trivially by state exposure. This security notion is independent from KIND security but is certainly nice to have for explicit authentication in key agreement. Besides, it is easy to achieve. We will also use it as a helper to prove KIND security: to reduce $\mathsf{C}^{\mathsf{P}}_{\mathsf{trivial\ forge}}$-cleanness to $\mathsf{C}^{\mathsf{P}}_{\mathsf{forge}}$-cleanness.

Let the adversary interact with the oracles $\mathsf{RATCH}, \mathsf{EXP}_{\mathsf{st}}$, $\mathsf{EXP}_{\mathsf{key}}$ in any order. For BARK to have unforgeability, we eliminate the trivial forgeries (as defined in Definition 12). The FORGE game is defined in Fig. 4.

Definition 15. (FORGE security). *Consider* $\mathsf{FORGE}^{\mathcal{A}}(1^\lambda)$ *game in Fig. 4 associated to the adversary* $\mathcal{A}$*. Let the advantage of* $\mathcal{A}$ *be the probability that the game outputs 1. We say that* BARK *is* FORGE-secure *if, for any PPT adversary, the advantage is negligible.*

```
Game $\mathsf{FORGE}^{\mathcal{A}}(1^\lambda)$
$\mathsf{Setup}(1^\lambda) \xrightarrow{\$} \mathsf{pp}$
$\mathsf{Initall}(1^\lambda, \mathsf{pp}) \xrightarrow{\$} (\mathsf{st}_A, \mathsf{st}_B, z)$
$(\mathsf{P}, \mathsf{upd}) \leftarrow \mathcal{A}^{\mathsf{RATCH}, \mathsf{EXP}_{\mathsf{st}}, \mathsf{EXP}_{\mathsf{key}}}(z)$
$\mathsf{RATCH}(\mathsf{P}, \mathsf{rec}, \mathsf{upd}) \to \mathsf{acc}$
if $\mathsf{acc} = \mathsf{false}$ then return 0
if $\mathsf{upd}$ is not a forgery for $\mathsf{P}$ then return 0
if $\mathsf{upd}$ is a trivial forgery for $\mathsf{P}$ then return 0
return 1
```

```
Game $\mathsf{RECOVER}^{\mathcal{A}}_{\mathsf{BARK}}(1^\lambda)$
$\mathsf{win} \leftarrow 0$
$\mathsf{Setup}(1^\lambda) \xrightarrow{\$} \mathsf{pp}$
$\mathsf{Initall}(1^\lambda, \mathsf{pp}) \xrightarrow{\$} (\mathsf{st}_A, \mathsf{st}_B, z)$
set all $\mathsf{sent}^*_*$ and $\mathsf{received}^*_*$ variables to $\emptyset$
$\mathsf{P} \leftarrow \mathcal{A}^{\mathsf{RATCH}, \mathsf{EXP}_{\mathsf{st}}, \mathsf{EXP}_{\mathsf{key}}}(z)$
if we can parse $\mathsf{received}^{\mathsf{P}}_{\mathsf{msg}} = (\mathsf{seq}_1, \mathsf{upd}, \mathsf{seq}_2)$ and $\mathsf{sent}^{\overline{\mathsf{P}}}_{\mathsf{msg}} = (\mathsf{seq}_3, \mathsf{upd}, \mathsf{seq}_4)$ with $\mathsf{seq}_1 \neq \mathsf{seq}_3$ (where $\mathsf{upd}$ is a single message and all $\mathsf{seq}_i$ are finite sequences of single messages) then
    $\mathsf{win} \leftarrow 1$
return $\mathsf{win}$
```

```
Game $\mathsf{PREDICT}^{\mathcal{A}}_{\mathsf{BARK}}(1^\lambda)$
$\mathsf{Setup}(1^\lambda) \xrightarrow{\$} \mathsf{pp}$
$\mathsf{Initall}(1^\lambda, \mathsf{pp}) \xrightarrow{\$} (\mathsf{st}_A, \mathsf{st}_B, z)$
$\mathsf{P} \leftarrow \mathcal{A}^{\mathsf{RATCH}, \mathsf{EXP}_{\mathsf{st}}, \mathsf{EXP}_{\mathsf{key}}}(z)$
$\mathsf{RATCH}(\mathsf{P}, \mathsf{send}) \to \mathsf{upd}$
if $\mathsf{upd} \in \mathsf{received}^{\overline{\mathsf{P}}}_{\mathsf{msg}}$ then return 1
return 0
```

Fig. 4. FORGE, RECOVER, and PREDICT games.

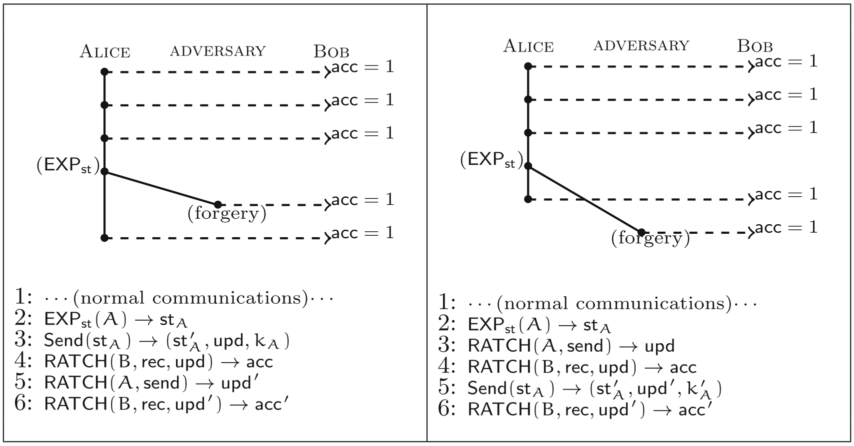
We can now justify why forgeries in the KIND game must be trivial for a BARK with unforgeability.

Theorem 16. *If a* BARK *is* FORGE*-secure, then* $(\mathsf{C}_{\mathsf{leak}} \wedge \mathsf{C}^{\mathsf{P}_{\mathsf{test}}}_{\mathsf{forge}})$-KIND-*security implies* $(\mathsf{C}_{\mathsf{leak}} \wedge \mathsf{C}^{\mathsf{P}_{\mathsf{test}}}_{\mathsf{trivial\ forge}})$-KIND-*security and* $(\mathsf{C}_{\mathsf{leak}} \wedge \mathsf{C}^{\mathsf{A}}_{\mathsf{forge}} \wedge \mathsf{C}^{\mathsf{B}}_{\mathsf{forge}})$-KIND-*security implies* $(\mathsf{C}_{\mathsf{leak}} \wedge \mathsf{C}^{\mathsf{A}}_{\mathsf{trivial\ forge}} \wedge \mathsf{C}^{\mathsf{B}}_{\mathsf{trivial\ forge}})$-KIND-*security.*

4.2 Recovery from Impersonation

A priori, it seems nice to be able to restore a secure state when a state exposure of a participant takes place. We show here that it is not a good idea.

Let $\mathcal{A}$ be an adversary playing the two games in Fig. 5. On the left strategy, $\mathcal{A}$ exposes A with an $\mathsf{EXP}_{\mathsf{st}}$ query (Step 2). Then, the adversary $\mathcal{A}$ impersonates A by running the Send algorithm on its own (Step 3). Next, the adversary $\mathcal{A}$ "sends" a message to B which is accepted due to correctness because it is generated with A's state. In Step 5, $\mathcal{A}$ lets the legitimate sender generate upd' by calling RATCH oracle. In this step, *if* security self-restores, then B accepts upd' which is sent by A, hence $\mathsf{acc}' = 1$. It is clear that the strategy shown on the left side in Fig. 5 is equivalent to the strategy shown on the right side of the same figure (which only switches Alice and the adversary who run the same algorithm). Hence, both lead to $\mathsf{acc}' = 1$ with the same probability p. The crucial point is that the forgery in the right-hand strategy becomes non-trivial, which implies that the protocol is not FORGE-secure. In addition to this, if such phenomenon occurs, we can make a KIND adversary passing the $\mathsf{C}_{\mathsf{leak}} \wedge \mathsf{C}^{\mathsf{P}_{\mathsf{test}}}_{\mathsf{trivial\ forge}}$ condition. Thus, we lose KIND-security. Consequently, security should *not* self-restore.



```
1: ··· (normal communications) ···
2: $\mathsf{EXP}_{\mathsf{st}}(A) \to \mathsf{st}_A$
3: $\mathsf{Send}(\mathsf{st}_A) \to (\mathsf{st}'_A, \mathsf{upd}, k_A)$
4: $\mathsf{RATCH}(B, \mathsf{rec}, \mathsf{upd}) \to \mathsf{acc}$
5: $\mathsf{RATCH}(A, \mathsf{send}) \to \mathsf{upd}'$
6: $\mathsf{RATCH}(B, \mathsf{rec}, \mathsf{upd}') \to \mathsf{acc}'$
```

```
1: ··· (normal communications) ···
2: $\mathsf{EXP}_{\mathsf{st}}(A) \to \mathsf{st}_A$
3: $\mathsf{RATCH}(A, \mathsf{send}) \to \mathsf{upd}$
4: $\mathsf{RATCH}(B, \mathsf{rec}, \mathsf{upd}) \to \mathsf{acc}$
5: $\mathsf{Send}(\mathsf{st}_A) \to (\mathsf{st}'_A, \mathsf{upd}', k'_A)$
6: $\mathsf{RATCH}(B, \mathsf{rec}, \mathsf{upd}') \to \mathsf{acc}'$
```

Fig. 5. Two recoveries succeeding with the same probability.

We define the RECOVER security notion with another game in Fig. 4. Essentially, in the game, we require the receiver P to accept some messages upd sent by the sender after the adversary makes successful forgeries in seq_1. We further use it as a second helper to prove KIND security with $\mathsf{C}_{\mathsf{ratchet}}$-cleanness.

Definition 17 (RECOVER **security**). *Consider* $\mathsf{RECOVER}^{\mathcal{A}}_{\mathsf{BARK}}(1^\lambda)$ *game in Fig. 4 associated to the adversary* $\mathcal{A}$*. Let the advantage of* $\mathcal{A}$ *in succeeding playing the game be* $\Pr(\mathsf{win} = 1)$*. We say that the ratcheted communication protocol is* RECOVER-secure*, if for any PPT adversary, the advantage is negligible.*

RECOVER-security iseasy to achieve using a collision-resistant hash function.

To be sure that no message was received before it was sent, we need the following security notion. In the PREDICT game, the adversary tries to make $\overline{\mathsf{P}}$ receive a message upd before it was sent by P.

Definition 18 (PREDICT **security**). *For the* $\mathsf{PREDICT}^{\mathcal{A}}_{\mathsf{BARK}}(1^\lambda)$ *game in Fig. 4, let* $\mathcal{A}$ *be an adversary. The advantage of* $\mathcal{A}$ *is the probability that* 1 *is returned. We say that the ratcheted communication protocol is* PREDICT-secure*, if for any adversary limited to a polynomially bounded number of queries, the advantage is negligible.*

Theorem 19. *If a* BARK *is* RECOVER*-secure,* PREDICT*-secure, and* $(\mathsf{C}_{\mathsf{leak}} \wedge \mathsf{C}^{A}_{\mathsf{forge}} \wedge \mathsf{C}^{B}_{\mathsf{forge}})$-KIND *secure, then it is* $(\mathsf{C}_{\mathsf{leak}} \wedge \mathsf{C}_{\mathsf{ratchet}})$-KIND *secure.*

5 Our BARK Protocol

We construct a BARK in Fig. 6. We use a public-key cryptosystem PKC, a digital signature scheme DSS, a one-time symmetric encryption Sym, and a collision-resistant hash function H. They are all defined in the full version [8]. First, we construct a naive signcryption SC from PKC and DSS by

$$
\begin{aligned}
\mathsf{SC.Enc}(\overbrace{\mathsf{sk_S}, \mathsf{pk_R}}^{\mathsf{st_S}}, \mathsf{ad}, \mathsf{pt}) &= \mathsf{PKC.Enc}(\mathsf{pk_R}, (\mathsf{pt}, \mathsf{DSS.Sign}(\mathsf{sk_S}, (\mathsf{ad}, \mathsf{pt})))) \\
\mathsf{SC.Dec}(\underbrace{\mathsf{sk_R}, \mathsf{pk_S}}_{\mathsf{st_R}}, \mathsf{ad}, \mathsf{ct}) &= (\mathsf{pt}, \sigma) \leftarrow \mathsf{PKC.Dec}(\mathsf{sk_R}, \mathsf{ct}) \ ; \\
&\quad\ \mathsf{DSS.Verify}(\mathsf{pk_S}, (\mathsf{ad}, \mathsf{pt}), \sigma) \ ? \ \mathsf{pt} \ : \ \perp
\end{aligned}
$$

Second, we extend SC to multi-key encryption called onion due to the multiple layers of keys. Third, we transform onion into a unidirectional ratcheting scheme uni. Finally, we design BARK. (See Fig. 6.)

The state of a participant is a tuple $\mathsf{st} = (\lambda, \mathsf{hk}, \mathsf{List_S}, \mathsf{List_R}, \mathsf{Hsent}, \mathsf{Hreceived})$ where hk is the hashing key, Hsent is the iterated hash of all sent messages, and Hreceived is the iterated hash of all received messages. We also have two lists $\mathsf{List_S}$ and $\mathsf{List_R}$. They are lists of states to be used for unidirectional communication: sending and receiving. Both lists are growing but entries are eventually erased. Thus, they can be compressed. (Typically, only the last entry is not erased.)

The idea is that the i^{th} entry of $\mathsf{List_S}$ for a participant P is associated to the i^{th} entry of $\mathsf{List_R}$ for its counterpart $\overline{\mathsf{P}}$. Every time a participant P sends a message, it creates a new pair of states for sending and receiving and sends the sending state to his counterpart $\overline{\mathsf{P}}$, to be used in the case $\overline{\mathsf{P}}$ wants to respond. If the same participant P keeps sending without receiving anything, he accumulates some receiving states this way. Whenever a participant $\overline{\mathsf{P}}$ who received many messages starts sending, he also accumulated many sending states. His message is sent using *all* those states in the uni.Send procedure. After sending, all but the last send state are erased, and the message shall indicate the erasures to the counterpart P, who shall erase corresponding receiving states accordingly. Our onion encryption needs to ensure $\mathcal{O}(n)$ complexity (so we cannot compose SC encryptions as ciphertext overheads would produce $\mathcal{O}(n^2)$ complexity). For that, we use a one-time symmetric encryption Sym using a key k in $\{0,1\}^{\mathsf{Sym.kl}}$. which is split into shares $k_1, \dots, k_n$. Each share is SC-encrypted in one state. Only the last state is updated (as others are meant to be erased).

The protocol is quite efficient when participants alternate their roles well, because the lists are often flushed to contain only one unerased state. It also becomes more secure due to ratcheting: any exposure has very limited impact. If there are unidirectional sequences, the protocol becomes less and less efficient due to the growth of the lists.

We state the security of our protocol below. Proofs are provided in the full version [8]. In the full version [8], we also show that our protocol does *not* offer $(\mathsf{C_{leak}} \wedge \mathsf{C_{forge}^{P_{test}}})$-KIND security.

Theorem 20. *We consider the* BARK *protocol from Fig. 6.*

- BARK *is correct.*
- BARK *is* PREDICT*-secure.*
- *If* H *is collision-resistant, then* BARK *is* RECOVER*-secure.*
- *If* DSS *is* SEF-OTCMA*-secure and* H *is collision-resistant, then* BARK *is* FORGE*-secure.*
- *If* PKC *is* IND-CCA*-secure and* Sym *is* IND-OTCCA*-secure, then* BARK *is* $(\mathsf{C_{leak}} \wedge \mathsf{C_{forge}^{A}} \wedge \mathsf{C_{forge}^{B}})$-KIND*-secure.*

```
$\mathsf{onion.Enc}(\mathsf{hk}, \mathsf{st}_S^1, \ldots, \mathsf{st}_S^n, \mathsf{ad}, \mathsf{pt})$
    pick $k_1, \ldots, k_n$ in $\{0,1\}^{\mathsf{Sym.kl}}$
    $k \leftarrow k_1 \oplus \cdots \oplus k_n$
    $\mathsf{ct}_{n+1} \leftarrow \mathsf{Sym.Enc}(k, \mathsf{pt})$
    $\mathsf{ad}_{n+1} \leftarrow \mathsf{ad}$
    for $i = n$ down to $1$ do
        $\mathsf{ad}_i \leftarrow \mathsf{H.Eval}(\mathsf{hk}, \mathsf{ad}_{i+1}, \mathsf{ct}_{i+1})$
        $\mathsf{ct}_i \leftarrow \mathsf{SC.Enc}(\mathsf{st}_S^i, \mathsf{ad}_i, k_i)$
    end for
    return $(\mathsf{ct}_1, \ldots, \mathsf{ct}_{n+1})$
```

```
$\mathsf{onion.Dec}(\mathsf{hk}, \mathsf{st}_R^1, \ldots, \mathsf{st}_R^n, \mathsf{ad}, \vec{\mathsf{ct}})$
    if $|\vec{\mathsf{ct}}| \neq n+1$ then return $\perp$
    parse $\vec{\mathsf{ct}} = (\mathsf{ct}_1, \ldots, \mathsf{ct}_{n+1})$
    $\mathsf{ad}_{n+1} \leftarrow \mathsf{ad}$
    for $i = n$ down to $1$ do
        $\mathsf{ad}_i \leftarrow \mathsf{H.Eval}(\mathsf{hk}, \mathsf{ad}_{i+1}, \mathsf{ct}_{i+1})$
        $\mathsf{SC.Dec}(\mathsf{st}_R^i, \mathsf{ad}_i, \mathsf{ct}_i) \rightarrow k_i$
        if $k_i = \perp$ then return $\perp$
    end for
    $k \leftarrow k_1 \oplus \cdots \oplus k_n$
    $\mathsf{pt} \leftarrow \mathsf{Sym.Dec}(k, \mathsf{ct}_{n+1})$
    return $\mathsf{pt}$
```

```
$\mathsf{uni.Init}(1^\lambda)$
    $\mathsf{SC.Gen}_S(1^\lambda) \xrightarrow{\$} (\mathsf{sk}_S, \mathsf{pk}_S)$
    $\mathsf{SC.Gen}_R(1^\lambda) \xrightarrow{\$} (\mathsf{sk}_R, \mathsf{pk}_R)$
    $\mathsf{st}_S \leftarrow (\mathsf{sk}_S, \mathsf{pk}_R)$
    $\mathsf{st}_R \leftarrow (\mathsf{sk}_R, \mathsf{pk}_S)$
    return $(\mathsf{st}_S, \mathsf{st}_R)$
```

```
$\mathsf{uni.Send}(1^\lambda, \mathsf{hk}, \vec{\mathsf{st}}_S, \mathsf{ad}, \mathsf{pt})$
    $\mathsf{SC.Gen}_S(1^\lambda) \xrightarrow{\$} (\mathsf{sk}'_S, \mathsf{pk}'_S)$
    $\mathsf{SC.Gen}_R(1^\lambda) \xrightarrow{\$} (\mathsf{sk}'_R, \mathsf{pk}'_R)$
    $\mathsf{st}'_S \leftarrow (\mathsf{sk}'_S, \mathsf{pk}'_R)$
    $\mathsf{st}'_R \leftarrow (\mathsf{sk}'_R, \mathsf{pk}'_S)$
    $\mathsf{pt}' \leftarrow (\mathsf{st}'_R, \mathsf{pt})$
    $\mathsf{onion.Enc}(1^\lambda, \mathsf{hk}, \vec{\mathsf{st}}_S, \mathsf{ad}, \mathsf{pt}') \rightarrow \vec{\mathsf{ct}}$
    return $(\mathsf{st}'_S, \vec{\mathsf{ct}})$
```

```
$\mathsf{uni.Receive}(\mathsf{hk}, \vec{\mathsf{st}}_R, \mathsf{ad}, \vec{\mathsf{ct}})$
    $\mathsf{onion.Dec}(\mathsf{hk}, \vec{\mathsf{st}}_R, \mathsf{ad}, \vec{\mathsf{ct}}) \rightarrow \mathsf{pt}'$
    if $\mathsf{pt}' = \perp$ then return $(\mathsf{false}, \perp, \perp)$
    parse $\mathsf{pt}' = (\mathsf{st}'_R, \mathsf{pt})$
    return $(\mathsf{true}, \mathsf{st}'_R, \mathsf{pt})$
```

```
$\mathsf{BARK.Setup}(1^\lambda)$
    $\mathsf{H.Gen}(1^\lambda) \xrightarrow{\$} \mathsf{hk}$
    return $\mathsf{hk}$
```

```
$\mathsf{BARK.Gen}(1^\lambda, \mathsf{hk})$
    $\mathsf{SC.Gen}_S(1^\lambda) \xrightarrow{\$} (\mathsf{sk}_S, \mathsf{pk}_S)$
    $\mathsf{SC.Gen}_R(1^\lambda) \xrightarrow{\$} (\mathsf{sk}_R, \mathsf{pk}_R)$
    $\mathsf{sk} \leftarrow (\mathsf{sk}_S, \mathsf{sk}_R)$
    $\mathsf{pk} \leftarrow (\mathsf{pk}_S, \mathsf{pk}_R)$
    return $(\mathsf{sk}, \mathsf{pk})$
```

```
$\mathsf{BARK.Init}(1^\lambda, \mathsf{hk}, \mathsf{sk}_P, \mathsf{pk}_{\overline{P}}, P)$
    parse $\mathsf{sk}_P = (\mathsf{sk}_S, \mathsf{sk}_R)$
    parse $\mathsf{pk}_{\overline{P}} = (\mathsf{pk}_S, \mathsf{pk}_R)$
    $\mathsf{st}_P^{\mathsf{send}} \leftarrow (\mathsf{sk}_S, \mathsf{pk}_R)$
    $\mathsf{st}_P^{\mathsf{rec}} \leftarrow (\mathsf{sk}_R, \mathsf{pk}_S)$
    $\mathsf{st}_P \leftarrow (\lambda, \mathsf{hk}, (\mathsf{st}_P^{\mathsf{send}}), (\mathsf{st}_P^{\mathsf{rec}}), \perp, \perp)$
    return $\mathsf{st}_P$
```

```
$\mathsf{BARK.Send}(\mathsf{st}_P)$
    pick $k$ at random in $\{0,1\}^{\mathsf{BARK.kl}}$
    parse $\mathsf{st}_P = (\lambda, \mathsf{hk}, (\mathsf{st}_P^{\mathsf{send},1}, \ldots, \mathsf{st}_P^{\mathsf{send},u}), (\mathsf{st}_P^{\mathsf{rec},1}, \ldots, \mathsf{st}_P^{\mathsf{rec},v}), \mathsf{Hsent}, \mathsf{Hreceived})$
    $\mathsf{uni.Init}(1^\lambda) \xrightarrow{\$} (\mathsf{st}_{S\mathsf{new}}, \mathsf{st}_P^{\mathsf{rec},v+1})$    ▷ append a new receive state to the $\mathsf{st}_P^{\mathsf{rec}}$ list
    $\mathsf{pt} \leftarrow (\mathsf{st}_{S\mathsf{new}}, k)$    ▷ $\mathsf{st}_{S\mathsf{new}}$ could be deleted immediately to avoid leaking
    take the smallest $i$ s.t. $\mathsf{st}_P^{\mathsf{send},i} \neq \perp$    ▷ $i = u - n$ if we had $n$ Receive since the last Send
    $\mathsf{uni.Send}(1^\lambda, \mathsf{hk}, (\mathsf{st}_P^{\mathsf{send},i}, \ldots, \mathsf{st}_P^{\mathsf{send},u}), \mathsf{Hsent}, \mathsf{pt}) \xrightarrow{\$} (\mathsf{st}_P^{\mathsf{send},u}, \vec{\mathsf{ct}})$    ▷ update $\mathsf{st}_P^{\mathsf{send},u}$
    $\mathsf{st}_P^{\mathsf{send},i}, \ldots, \mathsf{st}_P^{\mathsf{send},u-1} \leftarrow \perp$    ▷ flush the send state list: only $\mathsf{st}_P^{\mathsf{send},u}$ remains
    $\mathsf{upd} \leftarrow (\mathsf{Hsent}, \vec{\mathsf{ct}})$    ▷ $\vec{\mathsf{ct}}$ has $u - i + 2 (= n + 1)$ components
    $\mathsf{Hsent}' \leftarrow \mathsf{H.Eval}(\mathsf{hk}, \mathsf{upd})$
    $\mathsf{st}'_P \leftarrow (\lambda, \mathsf{hk}, (\mathsf{st}_P^{\mathsf{send},1}, \ldots, \mathsf{st}_P^{\mathsf{send},u}), (\mathsf{st}_P^{\mathsf{rec},1}, \ldots, \mathsf{st}_P^{\mathsf{rec},v+1}), \mathsf{Hsent}', \mathsf{Hreceived})$
    return $(\mathsf{st}'_P, \mathsf{upd}, k)$
```

```
$\mathsf{BARK.Receive}(\mathsf{st}_P, \mathsf{upd})$
    parse $\mathsf{st}_P = (\lambda, \mathsf{hk}, (\mathsf{st}_P^{\mathsf{send},1}, \ldots, \mathsf{st}_P^{\mathsf{send},u}), (\mathsf{st}_P^{\mathsf{rec},1}, \ldots, \mathsf{st}_P^{\mathsf{rec},v}), \mathsf{Hsent}, \mathsf{Hreceived})$
    parse $\mathsf{upd} = (h, \vec{\mathsf{ct}})$
    set $n + 1$ to the number of components in $\vec{\mathsf{ct}}$    ▷ the onion has $n$ layers
    if $h \neq \mathsf{Hreceived}$ then return $(\mathsf{false}, \mathsf{st}_P, \perp)$
    set $i$ to the smallest index such that $\mathsf{st}_P^{\mathsf{rec},i} \neq \perp$
    if $i + n - 1 > v$ then return $(\mathsf{false}, \mathsf{st}_P, \perp)$
    $\mathsf{uni.Receive}(\mathsf{hk}, (\mathsf{st}_P^{\mathsf{rec},i}, \ldots, \mathsf{st}_P^{\mathsf{rec},i+n-1}), \mathsf{Hreceived}, \vec{\mathsf{ct}}) \rightarrow (\mathsf{acc}, \mathsf{st}_P'^{\,\mathsf{rec},i+n-1}, \mathsf{pt})$
    if $\mathsf{acc} = \mathsf{false}$ then return $(\mathsf{false}, \mathsf{st}_P, \perp)$
    parse $\mathsf{pt} = (\mathsf{st}_P^{\mathsf{send},u+1}, k)$    ▷ a new send state is added in the list
    $\mathsf{st}_P^{\mathsf{rec},i}, \ldots, \mathsf{st}_P^{\mathsf{rec},i+n-2} \leftarrow \perp$    ▷ update $\mathsf{st}_P^{\mathsf{rec}}$ stage 1: clean up
    $\mathsf{st}_P^{\mathsf{rec},i+n-1} \leftarrow \mathsf{st}_P'^{\,\mathsf{rec},i+n-1}$    ▷ update $\mathsf{st}_P^{\mathsf{rec}}$ stage 2: update $\mathsf{st}_P^{\mathsf{rec},i+n-1}$
    $\mathsf{Hreceived}' \leftarrow \mathsf{H.Eval}(\mathsf{hk}, \mathsf{upd})$
    $\mathsf{st}'_P \leftarrow (\lambda, \mathsf{hk}, (\mathsf{st}_P^{\mathsf{send},1}, \ldots, \mathsf{st}_P^{\mathsf{send},u+1}), (\mathsf{st}_P^{\mathsf{rec},1}, \ldots, \mathsf{st}_P^{\mathsf{rec},v}), \mathsf{Hsent}, \mathsf{Hreceived}')$
    return $(\mathsf{acc}, \mathsf{st}'_P, k)$
```

Fig. 6. Our BARK protocol.

Consequently, due to Theorem 16, we deduce $(\mathsf{C}_{\mathsf{leak}} \wedge \mathsf{C}^{\mathsf{A}}_{\mathsf{trivial\ forge}} \wedge \mathsf{C}^{\mathsf{B}}_{\mathsf{trivial\ forge}})$-KIND-security. The advantage of treating $(\mathsf{C}_{\mathsf{leak}} \wedge \mathsf{C}^{\mathsf{A}}_{\mathsf{forge}} \wedge \mathsf{C}^{\mathsf{B}}_{\mathsf{forge}})$-KIND-security specifically is that we clearly separate the required security assumptions for SC.

Similarly, due to Theorem 19, we deduce $(\mathsf{C}_{\mathsf{leak}} \wedge \mathsf{C}_{\mathsf{ratchet}})$-KIND-security.

6 Conclusion

We studied the BARK protocols and security. For security, we marked three important security objectives: the BARK protocol should be KIND-secure; the BARK protocol should be resistant to forgery attacks (FORGE-security), and the BARK protocol should not self-heal after impersonation (RECOVER-security). By relaxing the cleanness notion in KIND-security, we designed a protocol based on an IND-CCA-secure cryptosystem and a one-time signature scheme. We used neither random oracle nor key-update primitives.

Acknowledgements. We thank Joseph Jaeger for his valuable comments to the first version of this paper. We thank Paul Rösler for insightful discussions and comments. We also owe to Andrea Caforio for his implementation results.

References

1. Alwen, J., Coretti, S., Dodis, Y.: The double ratchet: security notions, proofs, and modularization for the signal protocol. In: Ishai, Y., Rijmen, V. (eds.) EUROCRYPT 2019. LNCS, vol. 11476, pp. 129–158. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-17653-2_5. Full version: https://eprint.iacr.org/2018/1037.pdf
2. Bellare, M., Singh, A.C., Jaeger, J., Nyayapati, M., Stepanovs, I.: Ratcheted encryption and key exchange: the security of messaging. In: Katz, J., Shacham, H. (eds.) CRYPTO 2017. LNCS, vol. 10403, pp. 619–650. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-63697-9_21
3. Borisov, N., Goldberg, I., Brewer, E.: Off-the-record communication, or, why not to use PGP. In: Proceedings of the 2004 ACM Workshop on Privacy in the Electronic Society, WPES 2004, pp. 77–84. ACM, New York (2004)
4. Canetti, R., Krawczyk, H.: Analysis of key-exchange protocols and their use for building secure channels. In: Pfitzmann, B. (ed.) EUROCRYPT 2001. LNCS, vol. 2045, pp. 453–474. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-44987-6_28
5. Cohn-Gordon, K., Cremers, C., Dowling, B., Garratt, L., Stebila, D.: A formal security analysis of the signal messaging protocol. In: 2017 IEEE European Symposium on Security and Privacy (EuroS&P), pp. 451–466, April 2017
6. Cohn-Gordon, K., Cremers, C., Garratt, L.: On post-compromise security. In: 2016 IEEE 29th Computer Security Foundations Symposium (CSF), pp. 164–178, June 2016
7. Derler, D., Jager, T., Slamanig, D., Striecks, C.: Bloom filter encryption and applications to efficient forward-secret 0-RTT key exchange. In: Nielsen, J.B., Rijmen, V. (eds.) EUROCRYPT 2018. LNCS, vol. 10822, pp. 425–455. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-78372-7_14

8. Betül Durak, F., Vaudenay, S.: Bidirectional asynchronous ratcheted key agreement with linear complexity. https://eprint.iacr.org/2018/889.pdf
9. Günther, F., Hale, B., Jager, T., Lauer, S.: 0-RTT key exchange with full forward secrecy. In: Coron, J.-S., Nielsen, J.B. (eds.) EUROCRYPT 2017. LNCS, vol. 10212, pp. 519–548. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-56617-7_18
10. Jaeger, J., Stepanovs, I.: Optimal channel security against fine-grained state compromise: the safety of messaging. In: Shacham, H., Boldyreva, A. (eds.) CRYPTO 2018. LNCS, vol. 10991, pp. 33–62. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-96884-1_2. Full version: https://eprint.iacr.org/2018/553.pdf
11. Jost, D., Maurer, U., Mularczyk, M.: Efficient ratcheting: almost-optimal guarantees for secure messaging. In: Ishai, Y., Rijmen, V. (eds.) EUROCRYPT 2019. LNCS, vol. 11476, pp. 159–188. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-17653-2_6. Full version: https://eprint.iacr.org/2018/954.pdf
12. LaMacchia, B., Lauter, K., Mityagin, A.: Stronger security of authenticated key exchange. In: Susilo, W., Liu, J.K., Mu, Y. (eds.) ProvSec 2007. LNCS, vol. 4784, pp. 1–16. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-75670-5_1
13. Ohkubo, M., Suzuki, K., Kinoshita, S.: Cryptographic approach to “privacy-friendly” tags. In: RFID Privacy Workshop (2003)
14. Ohkubo, M., Suzuki, K., Kinoshita, S.: Efficient hash-chain based RFID privacy protection scheme. In: International Conference on Ubiquitous Computing (Ubicomp), Workshop Privacy: Current Status and Future Directions (2004)
15. Poettering, B., Rösler, P.: Towards bidirectional ratcheted key exchange. In: Shacham, H., Boldyreva, A. (eds.) CRYPTO 2018. LNCS, vol. 10991, pp. 3–32. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-96884-1_1. Full version: https://eprint.iacr.org/2018/296.pdf
16. Open Whisper Systems. Signal protocol library for Java/Android. GitHub repository (2017). https://github.com/WhisperSystems/libsignal-protocol-java
17. Unger, N., et al.: SoK: secure messaging. In: 2015 IEEE Symposium on Security and Privacy, pp. 232–249, May 2015
18. WhatsApp. Whatsapp encryption overview. Technical white paper (2016). https://www.whatsapp.com/security/WhatsApp-Security-Whitepaper.pdf