Bidirectional Asynchronous Ratcheted Key Agreement without Key-Update Primitives

F. Betül Durak and Serge Vaudenay

Ecole Polytechnique Fédérale de Lausanne (EPFL)
LASEC - Security and Cryptography Laboratory
Lausanne, Switzerland

Abstract. Following up mass surveillance and privacy issues, modern secure communication protocols now seek for more security such as forward secrecy and post-compromise security. They cannot rely on any assumption such as synchronization, predictable sender/receiver roles, or online availability. At EUROCRYPT 2017 and 2018, key agreement with forward secrecy and zero round-trip time (0-RTT) were studied. 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 0-RTT or without bidirectional communication. At CRYPTO 2018, it was done using key-update primitives, which involve hierarchical identity-based encryption (HIBE).

In this work, we define the bidirectional asynchronous ratcheted key agreement (BARK) with formal security notions. We provide a simple security model with a pragmatic approach and design the first secure BARK scheme not using key-update primitives. Our notion offers forward secrecy and post-compromise security. It is asynchronous, with random roles, and 0-RTT. It is based on a cryptosystem, a signature scheme, and a collision-resistant hash function family without key-update primitives or random oracles. Compared to previous protocols, ours is 100 to 1000 times faster. 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 that provides security for the users.

In bidirectional two-party secure communication, participants alternate their role as a *sender* and a *receiver*. Essentially, secure communication reduces to continuously exchanging keys, because each message requires a new key.

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 is undefined, meaning that participants use *random roles* (sender or receiver) without any synchronization. Namely, they could send messages at the same time. Being asynchronous, with 0-RTT, and random roles make the formalism more difficult and tedious.

Even though many systems were designed for the privacy of their users, they were rapidly faced with security vulnerabilities caused by the *compromises* of the participants' states. In this work, compromising a participant means to obtain some of its internal information. We will call it an *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 to decrypt past communication. This is called *forward secrecy*. Typically, forward secrecy is obtained by updating states with a one-way function $x \to H(x) \to H(H(x)) \to ...$ and deleting old entries. It is used, for instance, in RFID protocols [13, 14]. One mechanical technique to allow to move forward and to prevent from moving backward is to use a *ratchet*. In secure communication, ratcheting also includes the use of randomness in every state update so that a compromised state is not enough to decrypt future communication as well. 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 cut after active attacks.

Our goal is to obtain ratcheting security. To define it, we must exclude attacks which trivially exploit leakages. In this work, we adopt a very easy-to-understand rule: messages which are acknowledged by the legitimate receiver are considered safe (unless trivial passive attacks). This way, as soon as a sender is confirmed that his message was well received, he has strong guarantees that his message is safe and will remain so.

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 [19]. It is using Signal.

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.

At CRYPTO 2017, Bellare et al. [2] gave a secure ratcheting key exchange protocol. Their protocol is unidirectional and does not allow receiver exposure. They further construct secure communication (i.e. authentication and encryption) from key agreement and symmetric authenticated encryption.

At CRYPTO 2018, Poettering and Rösler [15] studied bidirectional asynchronous ratcheted key agreement and presented a protocol which is secure in the random oracle model. Their solution further relies on a hierarchical identity-based encryption (HIBE) but offers a stronger security than what we aim at, leaving the room to better protocols.

At the same conference, Jaeger and Stepanovs [10] did similar things 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 but send several messages, the depth grows up quite fast. Consequently, HIBE needs unbounded depth.

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

Two papers appeared after the first version of the current paper was released.

Jost, Maurer, and Mularczyk [1] designed another ratcheting protocol which has a *near-optimal security*, does not need HIBE, but has still 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. But 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 Poettering-Rösler [15] and Jaeger-Stepanovs [10].[1] One advantage of the Jost-Maurer-Mularczyk protocol [1] comes with the resilience with random coin leakage as discussed below.

Alwen, Coretti, and Dodis [11] 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 weakens forward secrecy, as it can only be obtained if adversaries passively let messages to be delivered. 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 "pragmatic". 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 (but are slower). The second protocol by Alwen, Coretti, and Dodis [11] uses asymmetric cryptography.

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

[1] This is only visible in the corrected version of the paper on eprint [10].

Table 1: Comparison of Protocols

	Security	Complexity		Coins leakage resilience
		alternating	accumulating	
Poettering-Rösler [15]	optimal	$\mathcal{O}(n)$	$\mathcal{O}(n^2)$	no
Jaeger-Stepanovs [10]	optimal	$\mathcal{O}(n)$	$\mathcal{O}(n^2)$	pre-send leakage, = state exposure
BARK [this paper]	sub-optimal	$\mathcal{O}(n)$	$\mathcal{O}(n)$	chosen coins, = state exposure
Jost-Maurer-Mularczyk [1]	near-optimal	$\mathcal{O}(n)$	$\mathcal{O}(n^2)$	post-send leakage
Alwen-Coretti-Dodis [11]	pragmatic	$\mathcal{O}(n)$	$\mathcal{O}(n)$	chosen coins, = state exposure

Our contributions. We give a definition for a bidirectional asynchronous key agreement (BARK) along with security properties. We give the appropriate 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 the 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 a helper to prove KIND-security. We finally construct a secure protocol. Our design choices are detailed below and compared to other papers.

1. **Simplicity**. Contrarily to previous work, we define KIND security in a very comprehensive way by moving all technicalities in 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 by 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 an 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 leaves to other applications which may leak for different reasons. In the beginning, we do not consider exposure of the random coins for simplicity. Later on, we show how to address random-coin-leakage resilience (with adversarially chosen random coins) in Section 3.3, by just taking Send operations with coin corruption as operations revealing both the generated key and the state.

3. **Slightly sub-optimal security**. Using the result from exposure allows the adversary to be quite active, e.g. by impersonating the exposed participant. However, the adversary is not allowed to use exposures to mount 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 Poettering-Rösler [15] and Jaeger-Stepanovs [10] which are considered of having optimal security (although it is not clear what optimality means, as we discuss in Appendix B) and than Jost-Maurer-Mularczyk [1] (which has near-optimal security), though, allowing lighter building blocks. Namely, we need no key-update primitives and have linear-time com-

plexity in terms of number of exchanged messages, even when the network is occasionally down. **This translates to a speed up factor of 100 to 1000 in implementations.** 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*.

4. **Sequence integrity**. We believe that duplex communication is reliably enforced by a lower level protocol. This solves packets loss by resend requests and to reconstruct the correct sequence order. What we only have to care for is when an adversary prevents the delivery of a message even though it has been requested several times. We made the choice to make the transmission of the next messages impossible under such attack. Contrarily, Alwen et al. [11] advocate 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 same keys. We define the stages *matching status, direct leakage, indirect leakage*. We aim to separate trivial attacks and forgeries from non-trivial cases with our definitions. Direct and indirect leakages define the times when the adversary can deduce the key generated due to the exposure of a participant who can either be the same participant (direct) or their counterpart (indirect). Such leakages cause trivial victory of the adversary.

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

Notations. We have two characters: Alice and Bob. Whenever we need an abbreviation, they are represented as A and B respectively. 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.

Structure of the paper. In Section 2, we define our BARK protocol along with correctness definition, and security of key indistinguishability, unforgeability, and unrecoverability. In Section 3, we give our BARK construction. Appendix A recalls definitions for underlying primitives. In Appendix C, we make some comments and comparison with the results of Bellare et al. [2], Poettering-Rösler [15], and Jaeger-Stepanovs [10].

2 Bidirectional Asynchronous Ratcheted Communication

2.1 BARK Definition and Correctness

A two-party ratcheted communication protocol consists of three protocols: Init, an initial state generation protocol between two communicating parties, called Alice and Bob; Send, a sender algorithm that is run when a participant wants to send a message; $\mathsf{Receive}$, a receiver algorithm that is run whenever a participant receives a message.

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

- $\mathsf{Init}(1^\lambda) \xrightarrow{\$} (\mathsf{st}_A, \mathsf{st}_B, z)$*: The initial state generation protocol* Init *inputs a security parameter* λ *and outputs a tuple* $(\mathsf{st}_A, \mathsf{st}_B, z)$ *which are initial states for both Alice and Bob and some public information* z.
- $\mathsf{Send}(\mathsf{st}_P) \xrightarrow{\$} (\mathsf{st}'_P, \mathsf{upd}, k)$*: The algorithm inputs a current state* st_P *for* $P \in \{A, B\}$. *It outputs a tuple* $(\mathsf{st}'_P, \mathsf{upd}, k)$ *with an updated state* st'_P*, a message* upd*, and a key* k.
- $\mathsf{Receive}(\mathsf{st}_P, \mathsf{upd}) \to (\mathsf{acc}, \mathsf{st}'_P, k)$*: The algorithm inputs* $(\mathsf{st}_P, \mathsf{upd})$ *where* $P \in \{A, B\}$. *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.* $(\mathsf{acc}, \mathsf{st}'_P, k)$.

A unidirectional asynchronous ratcheted key agreement (uniARK) *is a* BARK *in which Alice (called the* sender S*) only uses* Send *and Bob (called the* receiver R*) only uses* Receive.

In practice, it is convenient to consider Init algorithms which are *splittable*:

Definition 2 (Splittable Init). *We say that the* Init *algorithm of a* BARK *is splittable if there exists some algorithms* Gen_A*,* Gen_B*,* f_A*, and* f_B *such that* Init *is defined by*

$\mathsf{Init}(1^\lambda)$*:*
1: $\mathsf{Gen}_A(1^\lambda) \to (\mathsf{sk}_A, \mathsf{pk}_A)$
2: $\mathsf{Gen}_B(1^\lambda) \to (\mathsf{sk}_B, \mathsf{pk}_B)$
3: *pick* r
4: $\mathsf{st}_A \leftarrow (\mathsf{sk}_A, f_A(\mathsf{pk}_A, \mathsf{pk}_B, r))$
5: $\mathsf{st}_B \leftarrow (\mathsf{sk}_B, f_B(\mathsf{pk}_A, \mathsf{pk}_B, r))$
6: $z \leftarrow (\mathsf{pk}_A, \mathsf{pk}_B)$
7: ***return*** $(\mathsf{st}_A, \mathsf{st}_B, z)$

This way, private keys can be generated by their holders and there is no need to rely on an authority, except for authentication of pk_A and pk_B.

We consider bidirectional completely asynchronous communications. We can see, on Fig. 1, Alice and Bob running some sequences of Send and Receive operations without any prior agreement. Their time scale can be completely different. This means that Alice and Bob run algorithms in an asynchronous way. We define the scheduling by a sequence of users (Alice and Bob). Reading the sequence tells who executes a new step of the protocol. In our model, scheduling is controlled by the adversary. For the time being, we assume that the order of transmitted messages is preserved in each direction. If two messages arrive in different order or one was lost or replayed, it must be due to the attacks.

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 sequence of roles of Alice is not necessarily synchronized. Sending/receiving is refined by the $\mathsf{RATCH}(P, \mathsf{role}, [\mathsf{upd}])$ call in Fig. 2. In the correctness notion, sent messages by participants are buffered and delivered in the same order to the counterpart. So, both participants can send messages at the same time.

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 who generated upd. We formally define the correctness in Fig. 2. In gray, we put some instructions which are not necessary for the game itself. They define some variables that we will use later. $\mathsf{received}^P_{\mathsf{key}}$ (respectively $\mathsf{sent}^P_{\mathsf{key}}$) keeps a list of secret keys that are generated by P when running $\mathsf{Receive}$ (respectively, Send). Similarly, $\mathsf{received}^{\mathsf{P}}_{\mathsf{msg}}$ (respectively $\mathsf{sent}^{\mathsf{P}}_{\mathsf{msg}}$) keeps a list of upd information that are received (respectively sent) by P and accepted by $\mathsf{Receive}$. We stress that the $\mathsf{received}$ sequences only keep values for which $\mathsf{acc} = \mathsf{true}$. (This will be important in the security game.)

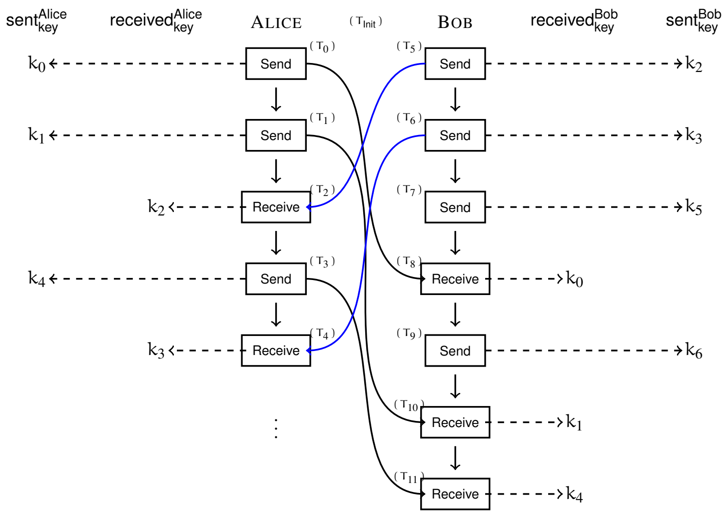


Fig. 1: The message exchange between Alice and Bob.

For two communicating parties Alice and Bob, we run Init to set up the states, and then run the correctness 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 run Send and send) or $(\mathsf{P}, \mathsf{rec})$ (saying that P must run $\mathsf{Receive}$ with whatever is received). 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.

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

The correctness implies that the decryption keys for the receiver have been generated same as encryption keys of the sender in the correct order. See Fig. 1 for the ordering of encryption/decryption keys, e.g. $\mathsf{sent}^{\mathsf{Alice}}_{\mathsf{key}} = \mathsf{received}^{\mathsf{Bob}}_{\mathsf{key}}$.

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

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

Oracle $\mathsf{RATCH}(\mathsf{P},\mathsf{rec},\mathsf{upd})$

$(\mathsf{acc},\mathsf{st}'_\mathsf{P},\mathsf{k}_\mathsf{P}) \leftarrow \mathsf{Receive}(\mathsf{st}_\mathsf{P},\mathsf{upd})$
if acc **then**
  $\mathsf{upd}_\mathsf{P} \leftarrow \mathsf{upd}$
  $\mathsf{st}_\mathsf{P} \leftarrow \mathsf{st}'_\mathsf{P}$
  append k_P to $\mathsf{received}^\mathsf{P}_\mathsf{key}$
  append upd_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 k_P to $\mathsf{sent}^\mathsf{P}_\mathsf{key}$
append upd_P to $\mathsf{sent}^\mathsf{P}_\mathsf{msg}$
return upd_P

Game Correctness(sched)

(for uniARK only) **if** $\exists i \quad (\mathsf{sched}_i = (\mathsf{A},\mathsf{rec})) \vee (\mathsf{sched}_i = (\mathsf{B},\mathsf{send}))$ **then** exit: adversary loses
set all sent^*_* and $\mathsf{received}^*_*$ variables to $\emptyset$
$\mathsf{Init}(1^\lambda) \xrightarrow{\$} (\mathsf{st}_\mathsf{A},\mathsf{st}_\mathsf{B},z)$
$i \leftarrow 0$
loop
  $i \leftarrow i+1$
  $(\mathsf{P},\mathsf{role}) \leftarrow \mathsf{sched}_i$
  if $\mathsf{role} = \mathsf{rec}$ **then**
    if no incoming message to P **then** exit: adversary loses
    pull upd from incoming messages to P
    $\mathsf{acc} \leftarrow \mathsf{RATCH}(\mathsf{P},\mathsf{rec},\mathsf{upd})$
    if $\mathsf{acc} = \mathsf{false}$ **then** exit: adversary wins
  else
    $\mathsf{upd} \leftarrow \mathsf{RATCH}(\mathsf{P},\mathsf{send})$
    push upd to incoming messages to $\overline{\mathsf{P}}$
  end if
  if $\mathsf{received}^\mathsf{A}_\mathsf{key}$ not prefix of $\mathsf{sent}^\mathsf{B}_\mathsf{key}$ **then** exit: adversary wins
  if $\mathsf{received}^\mathsf{B}_\mathsf{key}$ not prefix of $\mathsf{sent}^\mathsf{A}_\mathsf{key}$ **then** exit: adversary wins
end loop

Fig. 2: The correctness game.

to act both as a sender and as a receiver. 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.

RATCH. This is essentially the message exchange procedure. It is defined on 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.

EXP_st. The adversary can expose the state of Alice or Bob. It inputs $\mathsf{P} \in \{\mathsf{A},\mathsf{B}\}$ to the EXP_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 $b = 0$) or given as the last generated key of a participant P specified as input (if the challenge bit is $b = 1$). The oracle cannot be queried if no key was generated yet.

We specifically separate $\mathsf{EXP}_{\mathsf{key}}$ from $\mathsf{EXP}_{\mathsf{st}}$ as the key k generated by BARK will be used by the external process which may leak. Thus, $\mathsf{EXP}_{\mathsf{key}}$ can be more frequent than $\mathsf{EXP}_{\mathsf{st}}$, but will harm 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. We introduce these definitions as follows.

We use a notion of *time* and the value of the sequences $\mathsf{received}$ and sent at a given time. The security game executes instructions on a time scale and variables are updated. For all global variables v in the game such as $\mathsf{received}^{\mathsf{P}}_{\mathsf{msg}}$, k_{P}, or st_{P}, we denote by $v(t)$ the value of v at time t. For instance, $\mathsf{received}^{A}_{\mathsf{msg}}(t)$ is the sequence of upd which were received and accepted by A when running $\mathsf{Receive}$.

Definition 4 (Matching status). *At a given time* t, *we say that a participant* P *is in a* matching status *if there exist times* $\bar{t}$ *and* t' *such that 1.* $t' \leqslant t$, *2.* $\mathsf{received}^{\mathsf{P}}_{\mathsf{msg}}(t) = \mathsf{sent}^{\overline{\mathsf{P}}}_{\mathsf{msg}}(\bar{t})$, *and 3.* $\mathsf{received}^{\overline{\mathsf{P}}}_{\mathsf{msg}}(\bar{t}) = \mathsf{sent}^{\mathsf{P}}_{\mathsf{msg}}(t')$. *If this is the case, we say that time* t *for* P originates *from time* $\bar{t}$ *for* $\overline{\mathsf{P}}$.

The second condition clearly states that all the received (and accepted) upd information match the upd information sent by the counterpart of P, at some point in the past (at time $\bar{t}$), in the same order. The third 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 third condition is useful to prove that $k_{\mathsf{P}}(t) = k_{\overline{\mathsf{P}}}(\bar{t})$. This will be done in Lemma 8.

The key exchange literature often defines a notion of partnering which is simpler. What makes the notion more complicated here is the fact that we have asynchronous random roles.

An easy property of the notion of matching status is that if P is in a matching status at time t, then P is also in a matching status at any time $t_0 \leqslant t$. Similarly, if P is in a matching status at time t and t for P originates from $\bar{t}$ for $\overline{\mathsf{P}}$, then $\overline{\mathsf{P}}$ is in a matching status at time $\bar{t}$ and also at any time before. Note that although t originates from $\bar{t}$, which itself originates from t', we may have $t' \neq t$.

Definition 5 (Forgery). *Given a participant* P *in a game, we say that the* forgeries *in* $\mathsf{received}^{\mathsf{P}}_{\mathsf{msg}}$ *are* upd *messages* $\mathsf{upd}_1, \ldots, \mathsf{upd}_n$ *if there exist finite sequences of* upd *messages (possibly empty)* $\mathsf{seq}_0, \ldots, \mathsf{seq}_n$ *such that*

- $\mathsf{received}^{\mathsf{P}}_{\mathsf{msg}} = (\mathsf{seq}_0, \mathsf{upd}_1, \mathsf{seq}_1, \mathsf{upd}_2, \mathsf{seq}_2, \ldots, \mathsf{upd}_n, \mathsf{seq}_n)$;
- *for all* i, $(\mathsf{seq}_0, \mathsf{seq}_1, \ldots, \mathsf{seq}_{i-1})$ *is a prefix of* $\mathsf{sent}^{\overline{\mathsf{P}}}_{\mathsf{msg}}$;
- *for all* i, $(\mathsf{seq}_0, \mathsf{seq}_1, \ldots, \mathsf{seq}_{i-1}, \mathsf{upd}_i)$ *is not a prefix of* $\mathsf{sent}^{\overline{\mathsf{P}}}_{\mathsf{msg}}$.

Here, the comma operation "," is the concatenation of sequences and single messages upd_i *are taken as sequences of length* 1. *We call* upd_1 *as* P's first forgery.

Lemma 6. *If* P *is not in a matching status, either* P *or* $\overline{\mathsf{P}}$ *has received a forgery.*

Proof. If P did not receive a forgery, then $\mathsf{received}^{\mathsf{P}}_{\mathsf{msg}}$ is a prefix of $\mathsf{sent}^{\overline{\mathsf{P}}}_{\mathsf{msg}}$. Therefore, there exists a time $\bar{t}$ such that $\mathsf{received}^{\mathsf{P}}_{\mathsf{msg}}(t) = \mathsf{sent}^{\overline{\mathsf{P}}}_{\mathsf{msg}}(\bar{t})$. If P is not in matching status at time t, then $\mathsf{received}^{\overline{\mathsf{P}}}_{\mathsf{msg}}(\bar{t})$ cannot be a prefix of $\mathsf{sent}^{\mathsf{P}}_{\mathsf{msg}}(t)$. This implies that $\overline{\mathsf{P}}$ received a forgery due to Definition 5. □

A secure communication protocol needs such a "matching status" since it characterizes a normal execution of the protocol. More specifically, as we explained in previous section (and as it will become more clear later), "recovery from impersonation" cannot be allowed in BARK. A secure protocol should either enforce that both participants are always in matching status or make communication between them impossible.

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 7 (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} *one* corresponds *to the* j^{th} $\mathsf{RATCH}(\overline{\mathsf{P}},\mathsf{send})$ *call if* $i = j$ *and* P *is in matching status at the time of this* i^{th} *accepting* $\mathsf{RATCH}(\mathsf{P},\mathsf{rec},.)$ *call.*

Lemma 8. *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* $k_{\mathsf{P}} = k_{\overline{\mathsf{P}}}$.

Proof. If $\mathsf{RATCH}(\mathsf{P},\mathsf{rec},\mathsf{upd})$ and $\mathsf{RATCH}(\overline{\mathsf{P}},\mathsf{send})$ correspond to each other, then P is in matching status. We let t be the time of the $\mathsf{RATCH}(\mathsf{P},\mathsf{rec},\mathsf{upd})$ call and $\bar{t}$ be the time of the $\mathsf{RATCH}(\overline{\mathsf{P}},\mathsf{send})$. We make the sequence of all RATCH calls from P until time t and all RATCH calls from $\overline{\mathsf{P}}$ until time $\bar{t}$. By putting them in chronological order, thanks to the conditions of the matching status, we define a sequence sched, and the experiment runs as the correctness game. Due to correctness, the last calls generate the same key k. Hence, $k_{\mathsf{P}}(t) = k_{\overline{\mathsf{P}}}(\bar{t})$. □

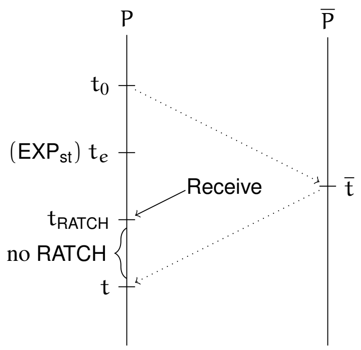
Definition 9 (Ratcheting period of P). *A maximal time interval during which there is no* $\mathsf{RATCH}(\mathsf{P},\mathsf{send})$ *call is called a* ratcheting period *of* P.

Consequently, a $\mathsf{RATCH}(\mathsf{P},\mathsf{send})$ call ends a ratcheting period for P and starts a new one. In Fig. 1, the time between T_1 and T_3 or the interval $T_5 - T_6$ are called ratcheting period of Alice and Bob respectively.

We now define the time 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 the case when the exposure was done on $\overline{\mathsf{P}}$ (indirect leakage).

Definition 10 (Direct leakage). *Let* t *be a time and* P *be a participant. We say that* $k_{\mathsf{P}}(t)$ *has a* direct leakage *if one of the following conditions is satisfied:*

- *There is an* $\mathsf{EXP_{key}}(\mathsf{P})$ *at a time* $\mathsf{t_e}$ *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* $\mathsf{t_e}$ *are the same.*
- P *is in a matching status and there exists* $\mathsf{t_0} \leqslant \mathsf{t_e} \leqslant \mathsf{t_{RATCH}} \leqslant \mathsf{t}$ *and* $\overline{\mathsf{t}}$ *such that time* t *originates from time* $\overline{\mathsf{t}}$*; time* $\overline{\mathsf{t}}$ *originates from time* $\mathsf{t_0}$*; there is one* $\mathsf{EXP_{st}}(\mathsf{P})$ *at time* $\mathsf{t_e}$*; there is one* $\mathsf{RATCH}(\mathsf{P},\mathsf{rec},.)$ *at time* $\mathsf{t_{RATCH}}$*; and there is no* $\mathsf{RATCH}(\mathsf{P},.,.)$ *between time* $\mathsf{t_{RATCH}}$ *and time* t.



In the first case, it is clear that $\mathsf{EXP_{key}}(\mathsf{P})$ gives $\mathsf{k_P}(\mathsf{t_e}) = \mathsf{k_P}(\mathsf{t})$. In the second case (in the figure[3]), the state which leaks from $\mathsf{EXP_{st}}(\mathsf{P})$ at time $\mathsf{t_e}$ allows to simulate all deterministic Receive (skipping all Send) and to compute the key $\mathsf{k_P}(\mathsf{t_{RATCH}}) = \mathsf{k_P}(\mathsf{t})$. The reason why we can 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 $\mathsf{k_P}(\mathsf{t})$.

Consider Fig. 1. Suppose t is in between time $\mathsf{T_3}$ and $\mathsf{T_4}$. According to our definition $\mathsf{P} = \mathsf{A}$ and the last RATCH call is at time $\mathsf{T_3}$. It is a Send, thus the second case cannot apply. The next RATCH call is at time $\mathsf{T_4}$. In this case, t has a direct leakage for Alice if there is a key exposure of Alice between $\mathsf{T_3}$ and $\mathsf{T_4}$.

Suppose now that $\mathsf{T_8} < \mathsf{t} < \mathsf{T_9}$. We have $\mathsf{P} = \mathsf{B}$, the last RATCH call is a Receive, it is at time $\mathsf{t_{RATCH}} = \mathsf{T_8}$, and t originates from time $\overline{\mathsf{t}} = \mathsf{T_0}$ which itself originates from the origin time $\mathsf{t_0} = \mathsf{T_{Init}}$ 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 $\mathsf{T_8}$. Indeed, with this last state exposure, the adversary can ignore all Send and simulate all Receive to derive $\mathsf{k_0}$.

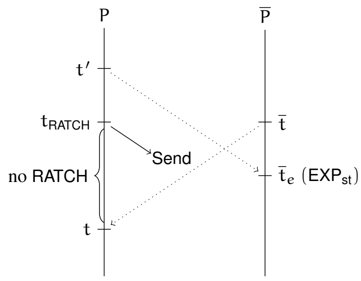
Definition 11 (Indirect leakage). *We consider a time* t *and a participant* P. *Let* $\mathsf{t_{RATCH}}$ *be the time of the last successful* RATCH *call and* role *be its input role. (We have* $\mathsf{k_P}(\mathsf{t_{RATCH}}) = \mathsf{k_P}(\mathsf{t})$.*) We say that* $\mathsf{k_P}(\mathsf{t})$ *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* $\mathsf{t'} \leqslant \mathsf{t_{RATCH}} \leqslant \mathsf{t}$ *and* $\overline{\mathsf{t}} \leqslant \overline{\mathsf{t}}_\mathsf{e}$ *such that* $\overline{\mathsf{P}}$ *is in a matching status at time* $\overline{\mathsf{t}}_\mathsf{e}$, t *originates from* $\overline{\mathsf{t}}$, $\overline{\mathsf{t}}_\mathsf{e}$ *originates from* $\mathsf{t'}$, *there is one* $\mathsf{EXP_{st}}(\overline{\mathsf{P}})$ *at time* $\mathsf{t_e}$, *and* $\mathsf{role} = \mathsf{send}$.

In the first case, $\mathsf{k_P}(\mathsf{t}) = \mathsf{k_P}(\mathsf{t_{RATCH}})$ 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 $\mathsf{EXP_{st}}(\overline{\mathsf{P}})$ then simulate all Receive with messages from P until the one sent at time $\mathsf{t_{RATCH}}$, ignoring all Send by $\overline{\mathsf{P}}$, to recover $\mathsf{k_P}(\mathsf{t})$.

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

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 Receive at time T_{10}, but t originates from the origin time $\overline{\mathsf{t}} = \mathsf{T}_{\mathsf{Init}}$. 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 $\overline{\mathsf{t}}_e$ (after time $\overline{\mathsf{t}} = 0$), originating from a time t' before time T_1, so $\overline{\mathsf{t}}_e < \mathsf{T}_{10}$ (second condition). In the latter case, the adversary can simulate 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 12 (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 has all information to compute* $\mathsf{k}_{\mathsf{P}}(\mathsf{t})$.

Proof. We use correctness, Lemma 8, and the explanations given after Def. 10 and Def. 11. $\square$

So far, we mostly focused on matching status cases but there could be situations with forgeries as well. We define trivial forgeries as follows.

Definition 13 (Trivial forgery). *We consider a first forgery* upd *received by* P *in a* $\mathsf{RATCH}(\mathsf{P},\mathsf{rec},\mathsf{upd})$ *call. Let* t *be the time just before this call. Let* $\overline{\mathsf{t}}$ *be a time such that* $\mathsf{received}^{\mathsf{P}}_{\mathsf{msg}}(\mathsf{t}) = \mathsf{sent}^{\overline{\mathsf{P}}}_{\mathsf{msg}}(\overline{\mathsf{t}})$*. If there is any* $\mathsf{EXP}_{\mathsf{st}}(\overline{\mathsf{P}})$ *call during the ratcheting period of* $\overline{\mathsf{P}}$ *which includes 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 5 in the KIND game on 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.)

- C^{P}_{forge}: P received no forgery until P has seen $\mathsf{upd}_{\mathsf{test}}$.
- C_{ratchet}: $\mathsf{upd}_{\mathsf{test}}$ was sent by a participant P, then received and accepted by $\overline{P}$, then some upd' was sent by $\overline{P}$, then upd' was received and accepted by P.

  (Here, P could be P_{test} or his counterpart. This accounts for the receipt of $\mathsf{upd}_{\mathsf{test}}$ being acknowledged by $\overline{P}$ through upd'.)
- $C_{\mathsf{noEXP}(R)}$: there is no $\mathsf{EXP}_{\mathsf{st}}(R)$ and no $\mathsf{EXP}_{\mathsf{key}}(R)$ query. (R is the receiver.)

Lemma 12 says that the adopted cleanness predicate C_{clean} must imply C_{leak} in all considered games. Otherwise, no security is possible. It is however not sufficient as it only covers trivial attacks with no forgeries.

C_{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, we do not expect more than the security notion from $C_{\mathsf{clean}} = C_{\mathsf{leak}} \wedge C_{\mathsf{ratchet}}$.

Bellare et al. [2] consider uniARK with $C_{\mathsf{clean}} = C_{\mathsf{leak}} \wedge C^{P_{\mathsf{test}}}_{\mathsf{trivial\ forge}} \wedge C_{\mathsf{noEXP}(R)}$. (See Appendix C.) Other papers like Poettering-Rösler [15] and Jaeger-Stepanovs [10] implicitly use $C_{\mathsf{clean}} = C_{\mathsf{leak}} \wedge C^{P_{\mathsf{test}}}_{\mathsf{trivial\ forge}}$ as cleanness predicate. They show that this is sufficient to build secure protocols but it is probably not the minimal cleanness predicate. Indeed, we know that *some* ways to make trivial forgeries (as defined) makes the adversary able to compute $k_{P_{\mathsf{test}}}(t_{\mathsf{test}})$ but there are some other ways not allowing the adversary to do so (see Appendix B). Hence, $C^{P_{\mathsf{test}}}_{\mathsf{trivial\ forge}}$ forbids more attacks than necessary.

Jost-Maurer-Mularczyk [1] excludes cases where $\overline{P}_{\mathsf{test}}$ received a (trivial) forgery then had an $\mathsf{EXP}_{\mathsf{st}}(\overline{P}_{\mathsf{test}})$ before receiving $\mathsf{upd}_{\mathsf{test}}$. Actually, they somehow use a cleanness predicate which is somewhere between $C_{\mathsf{leak}} \wedge C^{P_{\mathsf{test}}}_{\mathsf{trivial\ forge}}$ and $C_{\mathsf{leak}} \wedge C^{A}_{\mathsf{trivial\ forge}} \wedge C^{B}_{\mathsf{trivial\ forge}}$.

In our construction we use the predicate $C_{\mathsf{clean}} = C_{\mathsf{leak}} \wedge C^{A}_{\mathsf{forge}} \wedge C^{B}_{\mathsf{forge}}$. However, we define FORGE security (unforgeability) which implies that $(C_{\mathsf{leak}} \wedge C^{A}_{\mathsf{forge}} \wedge C^{B}_{\mathsf{forge}})$-$\mathsf{KIND}$ security and $(C_{\mathsf{leak}} \wedge C^{A}_{\mathsf{trivial\ forge}} \wedge C^{B}_{\mathsf{trivial\ forge}})$-$\mathsf{KIND}$ security are equivalent. (See Th. 17.) One drawback is that it forbids more than $(C_{\mathsf{leak}} \wedge C^{P_{\mathsf{test}}}_{\mathsf{trivial\ forge}})$-$\mathsf{KIND}$ security. The advantage is that we can achieve security without key-update primitives. We will prove in Th. 19 that this security is enough to achieve security with the predicate $C_{\mathsf{clean}} = C_{\mathsf{leak}} \wedge C_{\mathsf{ratchet}}$, thanks to $\mathsf{RECOVER}$-security. Thus, our cleanness notion is fair enough.

Definition 14 **(C_{clean}-KIND security).** *Let C_{clean} be a cleanness predicate. We consider the $\mathsf{KIND}^{\mathcal{A}}_{b,C_{\mathsf{clean}}}$ game of Fig. 3. We say that the ratcheted key agreement BARK is (q,T,ε)-C_{clean}-KIND-secure if for any adversary limited to q queries and time complexity T, the advantage*

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

of $\mathcal{A}$ in $\mathsf{KIND}^{\mathcal{A}}_{b,C_{\mathsf{clean}}}$ security game is bounded by ε.

```
Game KIND^A_{b,C_clean}
 1: Init(1^λ) →$ (st_A, st_B, z)
 2: set all sent*_* and received*_* variables to ∅
 3: set t_test, k_A, k_B to ⊥
 4: b' ← A^{RATCH,EXP_st,EXP_key,TEST}(z)
 5: if ¬C_clean then abort
 6: return b'

Oracle EXP_st(P)
 1: return st_P

Oracle TEST(P)
 1: if t_test ≠ ⊥ then abort        ▷ TEST was queried
 2: if k_P = ⊥ then abort
 3: t_test ← time, P_test ← P, upd_test ← upd_P
 4: if b = 1 then
 5:     return k_P
 6: else
 7:     return random {0,1}^{|k_P|}
 8: end if

Oracle EXP_key(P)
 1: return k_P
```

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

2.3 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 use it as a helper to prove KIND security: to reduce $\mathsf{C}^{\mathsf{P}}_{\text{trivial forge}}$-cleanness to $\mathsf{C}^{\mathsf{P}}_{\text{forge}}$-cleanness.

A first forgery is a upd received by a participant P making him lose his matching status. Let the adversary interact with our oracles 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 Def. 13). The FORGE game is defined in Fig. 4.

Definition 15 **(FORGE security).** *Consider* $\mathsf{FORGE}^{\mathcal{A}}$ *game in Fig. 4 associated to the adversary* $\mathcal{A}$*. Let the advantage of* $\mathcal{A}$ *in succeeding the attack in* $\mathsf{FORGE}^{\mathcal{A}}$ *game be the probability of succeeding the game. We say that* BARK *is* (q, T, ε)-FORGE-secure *if, for any adversary limited to* q *queries and time complexity* T*, the advantage is bounded by* ε*.*

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

Lemma 16. *Assume that* BARK *resists to* $\mathsf{FORGE}^{\mathcal{A}}$ *game. Let* $\mathcal{A}$ *be an adversary playing* $\mathsf{KIND}^{\mathcal{A}}_{b,\mathsf{C}_{\mathsf{clean}}}$ *game. For any* P *and* t*, if there exists no trivial forgery, the probability that* P *is not in matching status at a time* t *is negligible.*

Proof. It follows from Lemma 6 and the definition of the $\mathsf{FORGE}^{\mathcal{A}}$ game. □

Theorem 17. *If a* BARK *is* FORGE*-secure, then* $(\mathsf{C}_{\mathsf{leak}} \wedge \mathsf{C}^{\mathsf{P}_{\mathsf{test}}}_{\mathsf{forge}})$-KIND*-security implies* $(\mathsf{C}_{\mathsf{leak}} \wedge \mathsf{C}^{\mathsf{P}_{\mathsf{test}}}_{\text{trivial forge}})$-KIND*-security and* $(\mathsf{C}_{\mathsf{leak}} \wedge \mathsf{C}^{\mathsf{A}}_{\mathsf{forge}} \wedge \mathsf{C}^{\mathsf{B}}_{\mathsf{forge}})$-KIND*-security implies* $(\mathsf{C}_{\mathsf{leak}} \wedge \mathsf{C}^{\mathsf{A}}_{\text{trivial forge}} \wedge \mathsf{C}^{\mathsf{B}}_{\text{trivial forge}})$-KIND*-security.*

Proof. This is obvious, as FORGE-security implies no non-trivial forgery. □

Game $\mathsf{FORGE}^{\mathcal{A}}$

$\mathsf{Init}(1^\lambda) \xrightarrow{\$} (\mathsf{st}_A, \mathsf{st}_B, z)$
$(\mathsf{P}, \mathsf{upd}) \leftarrow \mathcal{A}^{\mathsf{RATCH}, \mathsf{EXP}_{\mathsf{st}}, \mathsf{EXP}_{\mathsf{key}}}(z)$
if one (or both) participants is NOT in a matching status **then** abort
$\mathsf{RATCH}(\mathsf{P}, \mathsf{rec}, \mathsf{upd}) \rightarrow \mathsf{acc}$
if $\mathsf{acc} = \mathsf{false}$ **then** abort
if P is in a matching status **then** abort
if upd is a trivial forgery for P **then** abort
the adversary wins

Game $\mathsf{RECOVER}^{\mathcal{A}}_{\mathsf{BARK}}$

$\mathsf{win} \leftarrow 0$
$\mathsf{Init}(1^\lambda) \xrightarrow{\$} (\mathsf{st}_A, \mathsf{st}_B, z)$
set all 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 upd is a single message and all seq_i are finite sequences of single messages) **then** $\mathsf{win} \leftarrow 1$
return win

Fig. 4: FORGE and RECOVER games.
(Oracle RATCH, $\mathsf{EXP}_{\mathsf{st}}$, $\mathsf{EXP}_{\mathsf{key}}$ are defined in Fig. 2 and Fig. 3.)

2.4 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 as shown 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 to generate upd' by calling RATCH oracle. In this step, *if* security self-restores, B accepts upd' which is sent by A. Hence, $\mathsf{acc}' = 1$ in the final step. 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 $C_{\mathsf{leak}} \wedge C^{\mathsf{P}_{\mathsf{test}}}_{\mathsf{trivial\ forge}}$ and $C_{\mathsf{leak}} \wedge C^{\mathsf{P}_{\mathsf{test}}}_{\mathsf{trivial\ forge}} \wedge C_{\mathsf{noEXP}(R)}$ conditions. Thus, we lose KIND-security.

In general, we believe it is not reasonable to allow recoveries from impersonation as it could serve as a discrete and temporary active attack and facilitate mass surveillance. For this purpose, we define the RECOVER security notion with another game. Essentially, in the game, we require the receiver P to accept some messages upd' sent by the sender after the adversary makes successful forgeries upd. We will further use it as a second helper to prove KIND security with C_{ratchet}-cleanness.

Definition 18 **(RECOVER security).** *Consider* $\mathsf{RECOVER}^{\mathcal{A}}_{\mathsf{BARK}}$ *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* (q, T, ε) RECOVER-secure, *if for any adversary limited to* q *queries and time complexity* T*, the advantage is bounded by* ε.

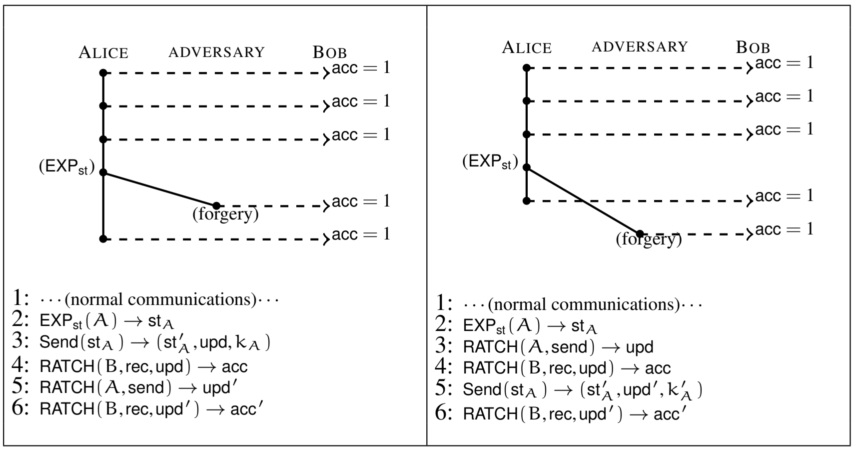


Fig. 5: Two recoveries succeeding with the same probability.

We will see that RECOVER-security is quite easy to achieve using a collision-resistant hash function.

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

Proof. Let us consider a $(C_{\mathsf{leak}} \wedge C_{\mathsf{ratchet}})$-KIND game in which C_{ratchet} holds. Let P be the participant who sent $\mathsf{upd}_{\mathsf{test}}$. Since $\mathsf{upd}_{\mathsf{test}}$ is a genuine message from P which is received by $\overline{P}$, the RECOVER security implies that $\overline{P}$ did not receive a forgery until it received $\mathsf{upd}_{\mathsf{test}}$ (except in negligible cases). So, $C^{\overline{P}}_{\mathsf{forge}}$ holds. Similarly, since P received a genuine upd' after seeing $\mathsf{upd}_{\mathsf{test}}$, P did not receive a forgery until then (except in negligible cases). So, C^P_{forge} holds, except in negligible cases. □

2.5 uniARK Implies KEM

We now prove that a weakly secure uniARK implies public key cryptography. Namely, we can construct a key encapsulation mechanism (KEM) out of it. We recall the KEM definition.

Definition 20 (KEM scheme). *A* KEM scheme KEM *consists of three algorithms: a key pair generation* $\mathsf{Gen}(1^\lambda) \xrightarrow{\$} (\mathsf{sk}, \mathsf{pk})$*, an encapsulation algorithm* $\mathsf{Enc}(\mathsf{pk}) \xrightarrow{\$} (k, \mathsf{ct})$*, and a decapsulation algorithm* $\mathsf{Dec}(\mathsf{sk}, \mathsf{ct}) \to k$*. It is correct if* $\Pr[\mathsf{Dec}(\mathsf{sk}, \mathsf{ct}) = k] = 1$ *when the keys are generated with* Gen *and* $\mathsf{Enc}(\mathsf{pk}) \to (k, \mathsf{ct})$.

We consider a uniARK which is KIND-secure for the following cleanness predicate:

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

(Note that R is never used.) This 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 21. *Given a* uniARK *protocol, we can construct a* KEM *with the following properties. The correctness of* uniARK *implies the correctness of* KEM. *The* $\mathsf{C_{weak}}$-KIND-*security of* uniARK *implies the* IND-CPA *security of* KEM.

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

$\mathsf{KEM.Gen} \xrightarrow{\$} (\mathsf{sk},\mathsf{pk})$**:** run $\mathsf{uniARK.Init} \xrightarrow{\$} (\mathsf{st_S},\mathsf{st_R},z)$ 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 IND-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:
1: $\mathsf{KEM.Gen} \xrightarrow{\$} (\mathsf{sk},\mathsf{pk})$
2: $\mathsf{KEM.Enc}(\mathsf{pk}) \xrightarrow{\$} (\mathsf{k},\mathsf{ct})$
3: **if** $b = 0$ **then** set k to random
4: $\mathcal{A}(\mathsf{pk},\mathsf{ct},\mathsf{k}) \xrightarrow{\$} b'$
5: **return** b'

Adversary $\mathcal{B}(z)$:
1: call $\mathsf{EXP_{st}}(S) \to \mathsf{pk}$
2: call $\mathsf{RATCH}(S,\mathsf{send}) \to \mathsf{ct}$
3: call $\mathsf{TEST}(S) \to \mathsf{k}$
4: run $\mathcal{A}(\mathsf{pk},\mathsf{ct},\mathsf{k}) \to b'$
5: **return** b'

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

3 A BARK Construction

3.1 Our BARK Protocol

We construct a BARK from a signcryption SC and a hash function H as on Fig. 6. Our construction is based on a *unidirectional asynchronous ratcheted communication with associated data* (uniARCAD), which itself is based on SC. The signcryption we use is a naive combination of a public-key cryptosystem and a digital signature scheme, as defined in Appendix A. The collision-resistant hash function is defined in Appendix A as well.

The Init protocol is splittable.

For each participant, the state is a tuple $\mathsf{st} = (\mathsf{hk},\mathsf{List_S},\mathsf{List_R},\mathsf{Hsent},\mathsf{Hreceived})$ 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 $\mathsf{List_S}$ resp. $\mathsf{List_R}$ of states. They are lists of states to be used for sending resp. receiving. Both lists are growing but start with erased entries. Thus, they can be compressed. (Typically, each list has only its last entry which 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 and sends the sending state to his counterpart $\overline{P}$, to be used in the case $\overline{P}$ wants to respond. If the same participant P keeps sending without receiving anything, he accumulates some receiving states this way. Whenever a participant $\overline{P}$ who received many messages starts sending, he also accumulated many sending states. His message is sent using *all* those states. The sent message is done by onion encapsulation using each remaining send state from List_S. Then, all but the last send state are erased, and the message shall indicate the erasures to the counterpart P, who shall erase receiving states accordingly. Unidirectional send operations in layers of onion encryption with $j < u$ need no state update (as the state is erased) while the first layer with $j = u$ needs a state update. This is why we added a flag in $\mathsf{uniARCAD.Send}$.

The protocol is quite efficient when participant 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. In practice, one might want to reuse a key k and a "symmetric ratchet" for sessions of unidirectional sequences. This will lower security a bit but would be perfectly in line with the current practice of "double ratchets".

We note that our protocol does *not* offer $(\mathsf{C}_{\mathsf{leak}} \wedge \mathsf{C}_{\mathsf{forge}}^{P_{\mathsf{test}}})$-$\mathsf{KIND}$ security due to the following attack:

1: $\mathsf{EXP}_{\mathsf{st}}(A) \to \mathsf{st}_A$
2: $\mathsf{EXP}_{\mathsf{st}}(B) \to \mathsf{st}_B$ ▷ this reveals $\mathsf{sk}_B^{\mathsf{rec},1}$ to be used later on
3: $\mathsf{RATCH}(B, \mathsf{send}) \to \mathsf{upd}_B$
4: $\mathsf{RATCH}(A, \mathsf{rec}, \mathsf{upd}_B) \to \mathsf{true}$
5: $\mathsf{RATCH}(A, \mathsf{send}) \to \mathsf{upd}$
6: $\mathsf{TEST}(A) \to k$
7: $\mathsf{Send}(\mathsf{st}_A) \to \mathsf{upd}_A$ ▷ this creates a trivial forgery
8: $\mathsf{RATCH}(B, \mathsf{rec}, \mathsf{upd}_A) \to \mathsf{true}$ ▷ this makes B out-of-sync and updates $\mathsf{sk}_B^{\mathsf{rec},1}$
9: $\mathsf{EXP}_{\mathsf{st}}(B) \to \mathsf{st}'_B$ ▷ this reveals $\mathsf{sk}_B^{\mathsf{rec},2}$ and $\mathsf{sk}_B^{\mathsf{rec},1}$ (updated)
10: use $\mathsf{sk}_B^{\mathsf{rec},1}$ (original) and $\mathsf{sk}_B^{\mathsf{rec},2}$ to decrypt upd
11: compare the result with k

Note that the trivial forgery is here to make the following $\mathsf{EXP}_{\mathsf{st}}(B)$ a non-trivial leakage for $\mathsf{sk}_B^{\mathsf{rec},2}$ ($\mathsf{sk}_B^{\mathsf{rec},1}$ is already known).

The attack is ruled out in the $(\mathsf{C}_{\mathsf{leak}} \wedge \mathsf{C}_{\mathsf{forge}}^{A} \wedge \mathsf{C}_{\mathsf{forge}}^{B})$-$\mathsf{KIND}$ security which does not allow forgeries until upd is received.

3.2 Security Proofs

We prove the security of BARK in this section.

Theorem 22 (Unrecoverability). *If H is a (T, ε)-collision-resistant hash function, then* BARK *on Fig. 6 is* (T, ε)-$\mathsf{RECOVER}$*-secure.*

Proof. Each upd sent must include the hash of the previous upd sent. We call them chained for this reason. If $(\mathsf{seq}_1, \mathsf{upd}, \mathsf{seq}_2)$ and $(\mathsf{seq}_3, \mathsf{upd}, \mathsf{seq}_4)$ are two validly chained list of messages with $\mathsf{seq}_1 \neq \mathsf{seq}_2$, we can easily see that $\mathsf{upd} = (n, h, \mathsf{onion})$ must include a collision h. This cannot happen, thanks to collision resistance. □

```
uniARCAD.Init(1^λ)
  SC.Gen_S(1^λ) →$ (sk_S, pk_S)
  SC.Gen_R(1^λ) →$ (sk_R, pk_R)
  st_S ← (sk_S, pk_R)
  st_R ← (sk_R, pk_S)
  z ← (pk_S, pk_R)
  return (st_S, st_R, z)
```

```
uniARCAD.Send(st_S, ad, pt, flag)
  parse st_S = (sk_S, pk_R)
  if flag then
      SC.Gen_S(1^λ) →$ (sk'_S, pk'_S)
      SC.Gen_R(1^λ) →$ (sk'_R, pk'_R)
      st'_S ← (sk'_S, pk'_R)
      st'_R ← (sk'_R, pk'_S)
  else
      st'_S, st'_R ← ⊥
  end if
  pt' ← (st'_R, pt)
  ct ← SC.Enc(sk_S, pk_R, ad, pt')
  return (st'_S, ct)
```

```
uniARCAD.Receive(st_R, ad, ct)
  parse st_R = (sk_R, pk_S)
  SC.Dec(sk_R, pk_S, ad, ct) → pt'
  if pt' = ⊥ then
      return (false, st_R, ⊥)
  end if
  parse pt' = (st'_R, pt)
  return (true, st'_R, pt)
```

```
BARK.Init(1^λ)
  uniARCAD.Init(1^λ) →$ (st_A^send, st_B^rec, z_{A→B})
  uniARCAD.Init(1^λ) →$ (st_B^send, st_A^rec, z_{B→A})
  H.Gen(1^λ) →$ hk
  st_A ← (hk, (st_A^send), (st_A^rec), ⊥, ⊥)
  st_B ← (hk, (st_B^send), (st_B^rec), ⊥, ⊥)
  z ← (z_{A→B}, z_{B→A})
  return (st_A, st_B, z)

BARK.Send(st_P)
  pick k at random
  parse st_P = (hk, (st_P^{send,1}, ..., st_P^{send,u}), (st_P^{rec,1}, ..., st_P^{rec,v}), Hsent, Hreceived)
  uniARCAD.Init(1^λ) →$ (st_Snew, st_P^{rec,v+1}, z)        ▷ append a new receive state to the st_P^rec list
  onion ← (st_Snew, k)                                      ▷ then, st_Snew is erased to avoid leaking
  take the smallest i s.t. st_P^{send,i} ≠ ⊥                 ▷ i = u − n if we had n Receive since the last Send
  for j = u down to i do                                    ▷ add encryption layers to onion and update st_P^send
      uniARCAD.Send(st_P^{send,j}, (u − j, Hsent), onion, j = u) →$ st_P^{send,j}, onion      ▷ update st_P^{send,j}
      if j < u then st_P^{send,j} ← ⊥                        ▷ flush the send state list: only st_P^{send,u} remains
  end for
  upd ← (u − i, Hsent, onion)                               ▷ the onion has u − i + 1 (= n + 1) layers
  Hsent' ← H.Eval(hk, upd)
  st'_P ← (hk, (st_P^{send,1}, ..., st_P^{send,u}), (st_P^{rec,1}, ..., st_P^{rec,v+1}), Hsent', Hreceived)
  return (st'_P, upd)

BARK.Receive(st_P, upd)
  parse st_P = (hk, (st_P^{send,1}, ..., st_P^{send,u}), (st_P^{rec,1}, ..., st_P^{rec,v}), Hsent, Hreceived)
  parse upd = (n, h, onion)                                 ▷ the onion has n + 1 layers
  if h ≠ Hreceived then return (false, st_P, ⊥)
  set i to the smallest index such that st_P^{rec,i} ≠ ⊥
  if i + n > v then return (false, st_P, ⊥)
  for j = i to i + n do                                     ▷ peel off onion and compute the next st_P^rec if accepted
      uniARCAD.Receive(st_P^{rec,j}, (i + n − j, Hreceived), onion) → (acc, st'_P^{rec,j}, onion)
      if acc = false then return (false, st_P, ⊥)
  end for
  parse onion = (st_P^{send,u+1}, k)                        ▷ a new send state is added in the list
  for j = i to i + n − 1 do                                 ▷ update st_P^rec stage 1: clean up
      st_P^{rec,j} ← ⊥
  end for                                                   ▷ n entries of st_P^rec were erased
  st_P^{rec,i+n} ← st'_P^{rec,i+n}                          ▷ update st_P^rec stage 2: update st_P^{rec,i+n}
  Hreceived' ← H.Eval(hk, upd)
  st'_P ← (hk, (st_P^{send,1}, ..., st_P^{send,u+1}), (st_P^{rec,1}, ..., st_P^{rec,v}), Hsent, Hreceived')
  return (acc, st'_P, k)
```

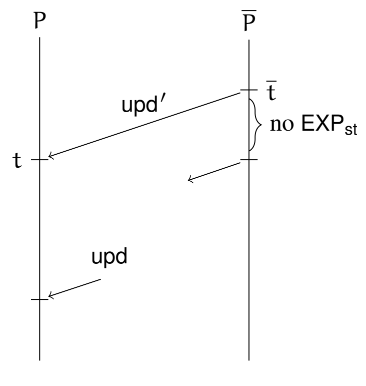
Fig. 6: Our BARK Protocol.

Theorem 23 (Unforgeability). *For any* q, T, ε*, assuming that* SC *is* (T', ε)-EF-OTCPA-*secure and* H *is a* (T, ε)*-collision-resistant hash function, then* BARK *on Fig. 6 is* $(q, T, q\varepsilon)$-FORGE-*secure. Here,* $T' = T + T_{\mathsf{Init}} + qT_{\mathsf{Send,Receive}}$ *where* T_{Init} *denotes a complexity upper bound of* Init *and* $T_{\mathsf{Send,Receive}}$ *denotes a complexity upper bound of both* Send *and* Receive.

Proof. We first prove that a forgery $\mathsf{upd} = (n, h, \mathsf{onion})$ for BARK corresponds to a forgery onion with $\mathsf{ad} = (n, h)$ for at least one instance of the uniARCAD protocol in the game. Then, we prove that we cannot forge a valid $(\mathsf{ad}, \mathsf{onion})$ pair in uniARCAD.

Let $\mathcal{A}$ be an adversary playing the FORGE game against BARK. We denote this game Γ. We assume without loss of generality that both participants are always in a matching status during Γ (otherwise, we make Γ abort as it will be the case in the FORGE game, eventually). Let m be the number of uniARCAD.Init calls during Γ. The first two are done in the initialization phase of Γ. All others are made by $\mathsf{RATCH}(., \mathsf{send})$ calls. We define m games $\Gamma_1, \ldots, \Gamma_m$ which simulate Γ. We can easily trace when the i^{th} uniARCAD.Init is run and when the two states it generates are used, evolve, and are erased. We denote those evolving states as st_S and st_R. The game Γ_i is playing the FORGE game against uniARCAD with those states (with a RATCH oracle updated in a straightforward manner in Γ_i because we deal with a uniARCAD instead of a BARK). It simulates the i^{th} uniARCAD.Init call by taking the initialized states in this game, and the uniARCAD.Send and uniARCAD.Receive by using some RATCH calls. Similarly, when st_S or st_R are needed in an $\mathsf{EXP}_{\mathsf{st}}$ call by Γ, we use the corresponding $\mathsf{EXP}_{\mathsf{st}}$ call in Γ_i. There is only one particular simulation: when $\mathsf{st}_{S_{\mathsf{new}}}$ is generated in BARK.Send, it must be onion-encrypted. Thus, we get it in Γ_i using $\mathsf{EXP}_{\mathsf{st}}(S)$. We call it the *extra* $\mathsf{EXP}_{\mathsf{st}}(S)$ *call*. The simulation is clearly perfect. We have to show that for any successful run of Γ, there exists at least one Γ_i which makes a non-trivial forgery in uniARCAD. We will prove below the FORGE-security of uniARCAD and therefore obtain the FORGE-security of BARK.

If we have a successful run releasing a forgery (P, upd) in Γ, we know that the forgery is not trivial in this game Γ. We denote $\mathsf{upd} = (n, h, \mathsf{onion})$.



In the first case, we assume that P successfully received a message upd' from $\overline{P}$ before upd. The receiving $\mathsf{RATCH}(P, \mathsf{rec}, \mathsf{upd}') \to \mathsf{true}$ call at some time t corresponds to a $\mathsf{RATCH}(\overline{P}, \mathsf{send}) \to \mathsf{upd}'$ call at some time $\bar{t}$. Since the forgery upd is non-trivial, this call starts a ratcheting session for $\overline{P}$ with no state exposure. The $\mathsf{RATCH}(\overline{P}, \mathsf{send})$ call at time $\bar{t}$ also defines some value u and some states $\mathsf{st}_{\overline{P}}^{\mathsf{send},u}$ and $\mathsf{st}_{P}^{\mathsf{rec},u}$. Let i be the index of the uniARCAD.Init call which initialized those states. This defines our game Γ_i of interest in which $\overline{P}$ is the sender S and P is the receiver R. After that corresponding $\mathsf{RATCH}(\overline{P}, \mathsf{send})$ at time $\bar{t}$, the list of send states of $\overline{P}$ is flushed and only $\mathsf{st}_{\overline{P}}^{\mathsf{send},u}$ remains (updated). After the $\mathsf{RATCH}(P, \mathsf{rec}, \mathsf{upd}')$ call at time t, $\mathsf{st}_{P}^{\mathsf{rec},u}$ will be the first active receive state in the list of P. The upd forgery must thus be first accepted by $\mathsf{uniARCAD.Receive}(\mathsf{st}_{\mathsf{P}}^{\mathsf{rec},u},(n,h),\mathsf{onion})$. If onion is not a forgery in the Γ_i game, it means that one $\mathsf{uniARCAD.Send}$ from a subsequent $\mathsf{RATCH}(\overline{\mathsf{P}},\mathsf{send})$ after time $\overline{t}$ which has some $\mathsf{uniARCAD.Send}(\mathsf{st}_{\overline{\mathsf{P}}}^{\mathsf{send},u},(n,h),\mathsf{onion}') \to (.,\mathsf{onion})$ with $h = \mathsf{Hsent}$. Since $\mathsf{Hsent} = \mathsf{H.Eval}(\mathsf{hk},\ldots,\mathsf{upd})$, we obtain a collision for H. Thanks to collision-resistance, we deduce that $(\mathsf{ad},\mathsf{onion})$ is a forgery in the Γ_i game. We can also observe that since it is non-trivial in Γ, it must be non-trivial in Γ_i as well. (Note that the $\mathsf{uniARCAD.Init}$ by P which generated the initial $\mathsf{st}_{\overline{\mathsf{P}}}^{\mathsf{send},u}$, required an extra $\mathsf{EXP}_{\mathsf{st}}(S)$ to onion-encrypt it in Γ_i but this $\mathsf{EXP}_{\mathsf{st}}$ does not make the forgery trivial as there was a subsequent ratcheting of S in Γ_i inside $\mathsf{RATCH}(\overline{\mathsf{P}},\mathsf{send})$ at time $\overline{t}$.) Therefore, Γ_i succeeds to forge in $\mathsf{uniARCAD}$.

In the second case, we assume that P never received anything from $\overline{\mathsf{P}}$. We proceed as before with $u = 1$. This state is initialized at the beginning of Γ so requires no extra $\mathsf{EXP}_{\mathsf{st}}(S)$. The proof is the same.

We now show that $\mathsf{uniARCAD}$ makes valid $(\mathsf{ad},\mathsf{upd})$ pairs unforgeable. To show this FORGE security, we can see that a first forgery consists of a pair $(\mathsf{ad},\mathsf{upd})$ which verifies with key pk_S. For each $\mathsf{SC.Gen}_S$ execution in the game, we construct a hybrid playing the $\mathsf{EF\text{-}OTCPA}$ game. This $\mathsf{EF\text{-}OTCPA}$ game is outsourcing the signing key sk_S and simulating Init and the RATCH calls in FORGE (hence the complexity of $T + T_{\mathsf{Init}} + qT_{\mathsf{Send,Receive}}$). We note that sk_S is kept in st_S and can only be used in signing with $\mathsf{SC.Enc}$ or in leaking with $\mathsf{EXP}_{\mathsf{st}}(S)$. So, we can fully outsource it in the $\mathsf{EF\text{-}OTCPA}$ game, with the exception in the leakage case. If there is any $\mathsf{EXP}_{\mathsf{st}}(S)$ to disclose st_S, we make the $\mathsf{EF\text{-}OTCPA}$ game abort. In the FORGE game, a first forgery which is non-trivial must correspond to a hybrid which succeeds in making a non-trivial forgery. Since it is non-trivial, there is no $\mathsf{EXP}_{\mathsf{st}}(S)$ call which is supposed to disclose sk_S. Hence, this hybrid playing $\mathsf{EF\text{-}OTCPA}$ wins. Due to the $\mathsf{EF\text{-}OTCPA}$ security of SC, those hybrids have a probability to succeed bounded by ε. Hence, forgeries must start by a trivial one, but for negligible cases. We deduce FORGE-security. □

Theorem 24 **(KIND Security).** *For any* q, T, ε*, assuming that* SC *is* (T', ε)-$\mathsf{IND\text{-}CCA}$-*secure, then* BARK *on Fig. 6 is* $(q, T, 2q\varepsilon)$-$(C_{\mathsf{leak}} \wedge C^{A}_{\mathsf{forge}} \wedge C^{B}_{\mathsf{forge}})$-$\mathsf{KIND}$-*secure. Here,* $T' = T + T_{\mathsf{Init}} + qT_{\mathsf{Send,Receive}}$ *where* T_{Init} *denotes a complexity upper bound of* Init *and* $T_{\mathsf{Send,Receive}}$ *denotes a complexity upper bound of both* Send *and* $\mathsf{Receive}$.

Due to Th. 17, Th. 23, and Th. 24, we deduce $(C_{\mathsf{leak}} \wedge C^{A}_{\mathsf{trivial\ forge}} \wedge C^{B}_{\mathsf{trivial\ forge}})$-$\mathsf{KIND}$-security. The advantage of treating $(C_{\mathsf{leak}} \wedge C^{A}_{\mathsf{forge}} \wedge C^{B}_{\mathsf{forge}})$-$\mathsf{KIND}$-security specifically is that we clearly separate the required security assumptions for SC.

Due to Th. 19, Th. 22, and Th. 24, we deduce $(C_{\mathsf{leak}} \wedge C_{\mathsf{ratchet}})$-$\mathsf{KIND}$-security.

Proof. We take a KIND game which we denote by Γ. The idea is that we will identify which keys generated by $\mathsf{SC.Gen}_R$ are safe and apply the $\mathsf{IND\text{-}CCA}$ reduction to whatever they encrypt. This way, we hope that the key k which is tested by TEST will be replaced by a random one and never used in a distinguishable way. The difficulties are

- to identify which keys are safe;
- to get rid of a safe sk_R (except for decryption) to apply the $\mathsf{IND\text{-}CCA}$ game;
- to see the connection between C_{clean} and the notion of safe key.

We number each use of $\mathsf{SC.Gen}_R$ with an index j. All indices are set in chronological order. For each j, we define a list $i_{j,1},\ldots,i_{j,\ell_j}$ of length ℓ_j. The j^{th} run of $\mathsf{SC.Gen}_R$ is either done on Step 2 in $\mathsf{uniARCAD.Init}$ (called either by $\mathsf{ARCAD.Init}$ or $\mathsf{ARCAD.Send}$) or on Step 4 in $\mathsf{uniARCAD.Send}$ (called by $\mathsf{ARCAD.Send}$). If it is done in $\mathsf{uniARCAD.Init}$, we set $\ell_j = 0$. Actually, the receive decryption key sk_R which is generated by $\mathsf{SC.Gen}_R$ stays local on the participant which generated it in $\mathsf{BARK.Send}$ (or $\mathsf{BARK.Init}$). Otherwise, sk_R is generated during a $\mathsf{uniARCAD.Send}$ called by $\mathsf{BARK.Send}$ and it will be encrypted in an onion to be sent to the other participant. There is at least one encryption in the generating $\mathsf{uniARCAD.Send}$ (Step 11) but it may be followed by more encryptions in the onion. We let $i_{j,1},\ldots,i_{j,\ell_j}$ be the indices of the $\mathsf{SC.Gen}_R$ runs which generated the keys which are needed to onion-decrypt sk_R. (If some keys were not generated by a $\mathsf{SC.Gen}_R$ run of the game, they are not listed.) We note that those indices are all lower than j, due to the chronological order.

In a game, for each j we define a flag NoEXP_j. The j^{th} decryption key sk_R generated by the j^{th} run of $\mathsf{SC.Gen}_R$ appears in some $\mathsf{st}^{\mathsf{rec}}$ in st_A or st_B. If there is no oracle call $\mathsf{EXP}_{\mathsf{st}}(P)$ at a time when st_P includes sk_R, we set NoEXP_j to true. Otherwise, we set it to false. Hence, NoEXP_j indicates if the j^{th} key sk_R is revealed by some $\mathsf{EXP}_{\mathsf{st}}$. One problem is that NoEXP_j can only be determined for sure after the key is updated or erased by a successful $\mathsf{BARK.Receive}$.

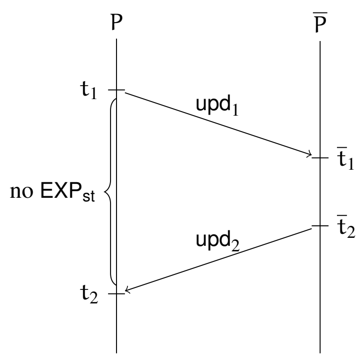
For each j, if $\ell_j = 0$, we define $\mathsf{SafeKey}_j = \mathsf{NoEXP}_j$. Otherwise, we define recursively safe keys as those which are not exposed and which are encrypted by at least one safe key:

$$\mathsf{SafeKey}_j = \left(\mathsf{SafeKey}_{i_{j,1}} \vee \cdots \vee \mathsf{SafeKey}_{i_{j,\ell_j}}\right) \wedge \mathsf{NoEXP}_j$$

This is well defined because the indices $i_{j,1},\ldots,i_{j,\ell_j}$ are all lower than j.

To understand which keys are safe, let us consider some RATCH calls:

- $\mathsf{RATCH}(P,\mathsf{send}) \to \mathsf{upd}_1$ at time t_1 (some sk_R is generated by $\mathsf{uniARCAD.Init}$),
- $\mathsf{RATCH}(\overline{P},\mathsf{rec},\mathsf{upd}_1) \to \mathsf{true}$ at time $\overline{t}_1$,
- $\mathsf{RATCH}(\overline{P},\mathsf{send}) \to \mathsf{upd}_2$ at time $\overline{t}_2 > \overline{t}_1$,
- $\mathsf{RATCH}(P,\mathsf{rec},\mathsf{upd}_2) \to \mathsf{true}$ at time $t_2 > t_1$.



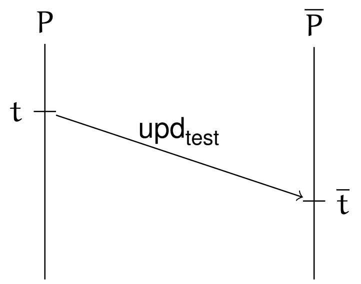
This is a round-trip $P \to \overline{P} \to P$. We assume that there is no $\mathsf{EXP}_{\mathsf{st}}(P)$ between t_1 and t_2. Hence, the new receive key sk_R generated by P in $\mathsf{uniARCAD.Init}$ at time t_1 stays in P. It is used to decrypt upd_2 at time t_2 then destroyed (actually, sk_R is updated into another key generated by $\overline{P}$). As there is no $\mathsf{EXP}_{\mathsf{st}}(P)$ to reveal sk_R between time t_1 and t_2, this key sk_R is safe. As long as no $\mathsf{EXP}_{\mathsf{st}}(P)$ reveals them, the key generated by $\overline{P}$ in $\mathsf{uniARCAD.Send}$ at time $\overline{t}_2$ to update sk_R at time t_2 (and in subsequent $\mathsf{RATCH}(\overline{P},\mathsf{send})$ as long as there is no $\mathsf{RATCH}(\overline{P},\mathsf{rec},.)$) is also safe as it is safely encrypted for the decryption key sk_R.

We define hybrid games Γ_j starting from $\Gamma_0 = \Gamma$. In those games, there is a flag bad which is set to false at the beginning. Some st^R states in st_A or st_B will include some decryption keys sk_R which will be replaced in hybrid games by random values and clearly marked as such. If any $\mathsf{EXP}_{\mathsf{st}}$ call reveals a state which includes such marked key, the flag bad is set to true and the game aborts.

Given Γ_{j-1}, we look at the j^{th} run of $\mathsf{SC.Gen}_R$. We let pk_R be the encryption key and sk_R be the decryption key. We compute the flag NoEXP_j and $\mathsf{SafeKey}_j$ in Γ_{j-1}. If $\mathsf{SafeKey}_j = \mathsf{false}$, we set $\Gamma_j = \Gamma_{j-1}$. Otherwise, once generated, we replace sk_R by a well-marked random value, but we use the right sk_R when it is needed in a $\mathsf{SC.Dec}$ execution. If the key sk_R is not onion-encrypted, the two games give exactly the same result as $\mathsf{NoEXP}_j = \mathsf{true}$ and sk_R is only used for decryption. If the key sk_R is onion-encrypted, since $\mathsf{SafeKey}_j = \mathsf{true}$, there must be one index $j_{i_{j,m}}$ such that $\mathsf{SafeKey}_{j_{i_{j,m}}} = \mathsf{true}$. We can use the $\mathsf{IND\text{-}CCA}$ game with the key of index $j_{i_{j,m}}$ to show that the encryption of the real sk_R or some random value are indistinguishable, up to an advantage of ε. The probability that bad becomes true in Γ_{j-1} and Γ_j cannot differ by more than ε as well.

Eventually, we obtain a game Γ_q in which bad is true with negligible probability and giving an outcome which is indistinguishable from Γ. In Γ_q, all keys sk_R which are safe are marked and replaced by a random value, so only used for decryption. Hence, we can apply the $\mathsf{IND\text{-}CCA}$ game for any of the safe keys.

Now, we can analyze what happens if the key k tested with $\mathsf{TEST}(P_{\mathsf{test}})$ at time t_{test} is replaced by a random one, when the cleanness property of the KIND game is satisfied.



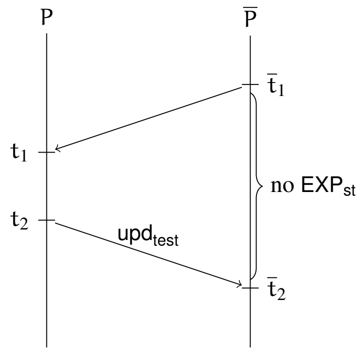
$P_{\mathsf{test}} \in \{P, \overline{P}\}$

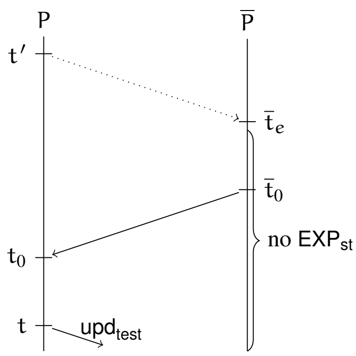
First of all, we note that the key $k_{\mathsf{test}} = k_{P_{\mathsf{test}}}(t_{\mathsf{test}})$ is made on P_{test} either by $\mathsf{BARK.Send}$ together with $\mathsf{upd}_{\mathsf{test}}$ (so generated by this algorithm), or by $\mathsf{BARK.Receive}$ so transmitted before through $\mathsf{upd}_{\mathsf{test}}$. Due to the $C^A_{\mathsf{forge}} \wedge C^B_{\mathsf{forge}}$ cleanness condition, $\mathsf{upd}_{\mathsf{test}}$ is not a forgery. So, k_{test} is always originally made by a $\mathsf{BARK.Send}$ which generated $\mathsf{upd}_{\mathsf{test}}$. In what follows we denote by P the participant who runs this $\mathsf{BARK.Send}$ and by t the time when this execution terminates. Let $\bar{t}$ be the time when $\overline{P}$ ends the reception of $\mathsf{upd}_{\mathsf{test}}$ (let $\bar{t} = \infty$ if it never receives it). Hence, k_{test} is generated by P and somehow sent to $\overline{P}$. Note that P_{test} may be P (so $k_{\mathsf{test}} = k_P(t)$) or $\overline{P}$ (so $k_{\mathsf{test}} = k_{\overline{P}}(\bar{t})$). We stress that thanks to the $C^A_{\mathsf{forge}} \wedge C^B_{\mathsf{forge}}$ assumption and Lemma 6, P is in a matching status at time t and $\overline{P}$ is in a matching status at time $\bar{t}$.

Clearly, k_{test} is not revealed by any $\mathsf{EXP}_{\mathsf{key}}$ due to the assumption that there is *no direct or indirect leakage*. Hence, $\mathsf{EXP}_{\mathsf{key}}$ never uses k_{test}. So, k_{test} is only used during onion encryption in $\mathsf{upd}_{\mathsf{test}}$ and by TEST.

Now, we can look at which flow of onion encryption followed the k_{test} generation to reach the receiver $\overline{P}$, with the *cleanness assumption*. The onion encryption is done with some keys defined in $\mathsf{st}_P^{\mathsf{send},u}, \mathsf{st}_P^{\mathsf{send},u-1}, \ldots, \mathsf{st}_P^{\mathsf{send},i}$. We show below that k_{test} is transmitted with at least one safe encryption (in the sense of the $\mathsf{SafeKey}_j$ flag). Hence, we can use the IND-CCA game for this safe encryption. We deduce that k_{test} is only used by TEST, so indistinguishable from random. We obtain KIND security. Therefore, what remains to be proven is that k is encrypted by at least one safe encryption.

We start with the $\bar{t} < \infty$ case: $\overline{P}$ receives $\mathsf{upd}_{\mathsf{test}}$ at some point. We recall that $\overline{P}$ must be in a matching status, due to the above discussion. Hence, both P and $\overline{P}$ have k_{test} and P_{test} is one or the other. Due to the C_{leak} hypothesis, $\overline{P}$ has no direct leakage at time $\bar{t}$. (This is straightforward if $P_{\mathsf{test}} = \overline{P}$, and this comes from the *first condition of indirect leakage* if $P_{\mathsf{test}} = P$.) Since $\overline{P}$ receives $\mathsf{upd}_{\mathsf{test}}$, the condition of no direct leakage implies that either there is no prior $\mathsf{EXP}_{\mathsf{st}}$ or there is a round-trip communication $\overline{P} \to P \to \overline{P}$ in between the last $\mathsf{EXP}_{\mathsf{st}}$ and time $\bar{t}$, hence, a message sent by $\overline{P}$ after the last $\mathsf{EXP}_{\mathsf{st}}$ and received by P before time t. Due to our previous analysis on this round trip, this means that $\mathsf{upd}_{\mathsf{test}}$ was encrypted with a safe encryption.





If now $\bar{t} = \infty$ ($\overline{P}$ never receives upd; so $P_{\mathsf{test}} = P$) and there are some $\mathsf{EXP}_{\mathsf{st}}(\overline{P})$ queries, due to the *no forgery assumption*, $\overline{P}$ stays in a matching status originating from a time prior to t. The *second condition of no indirect leakage* on P at time t implies that if $\bar{t}_e$ denotes the time of the latest $\mathsf{EXP}_{\mathsf{st}}(\overline{P})$ and t' denotes the time when it originates from, then there is a $\mathsf{RATCH}(\overline{P}, \mathsf{send}) \to \mathsf{upd}$ at a time $\bar{t}_0$ after time $\bar{t}_e$ and a corresponding $\mathsf{RATCH}(P, \mathsf{rec}, \mathsf{upd})$ at a time t_0 between time t' and time t. The uniARCAD.Send in the onion sent at time $\bar{t}_0$ generates a safe key which is used to encrypt the next sent upd from P, and $\mathsf{upd}_{\mathsf{test}}$ as well.

We now consider the case $\bar{t} = \infty$ with no $\mathsf{EXP}_{\mathsf{st}}(\overline{P})$ query. With a similar analysis as before, the last reception key generated for $\overline{P}$ is safe. So, $\mathsf{upd}_{\mathsf{test}}$ is safely encrypted. $\square$

3.3 Addressing Random Coin Corruption

Assuming that an adversary can control the random coins which are selected during a Send operation, the benefit of ratcheting is lost. In our security game, we could add a new option to the oracle RATCH which does the same as RATCH with role send but with an extra input which is the sequence of random coins to be used by Send. By treating those RATCH calls as if they were followed by $\mathsf{EXP}_{\mathsf{st}}$ and $\mathsf{EXP}_{\mathsf{key}}$ at the same time, we make sure that our security notion would not change and normal RATCH with role send would be healing.

Oracle $\mathsf{RATCH}(\mathrm{P}, \mathsf{send}, r)$
1: $(\mathsf{st_P}, \mathsf{upd_P}, \mathsf{k_P}) \leftarrow \mathsf{Send}(\mathsf{st_P}; r)$
2: $\mathsf{EXP_{key}}(\mathrm{P})$
3: $\mathsf{EXP_{st}}(\mathrm{P})$
4: **return** $\mathsf{upd_P}$

Otherwise, we would need to add conditions in the C_{leak} predicate by taking into account the Send queries with coin leakage. We can see that the proof of our BARK protocol still works in this setting. We only need to add a clause on the definition of $\mathsf{SafeKey_j}$: that the considered $\mathsf{SC.Gen_R}$ did not leak with coins in the Send query which run $\mathsf{SC.Gen_R}$.

It is quite normal to assume $\mathsf{EXP_{key}}$ is done as the generated key depends on freshly flipped coins. As for $\mathsf{EXP_{st}}$, this is less clear. Actually, Jost et al. [1] have a subtle protocol making sure that corrupted coins do not imply leaking the state. So far, no other protocol offers such property.

4 Conclusion

We studied the BARK security. For this, we marked three important security objectives: the BARK protocol should be KIND-secure; the BARK protocol should resist to unforgeability (FORGE-security). Moreover, 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 no random oracle nor key-update primitives. We implemented BARK and competing protocols (Poettering-Rösler [15], Jaeger-Stepanovs [10], and Jost-Maurer-Mularczyk [1]; we did not implement yet Alwen-Coretti-Dodis [11] which play in another category). We observed a speed up factor between 100 and 1000, depending on how messages are exchanged (namely, alternating or unidirectional).

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

References

1. Joël Alwen, Sandro Coretti, and Yevgeniy Dodis. The double ratchet: Security notions, proofs, and modularization for the signal protocol. Available at: `https://eprint.iacr.org/2018/1037.pdf`.
2. Mihir Bellare, Asha Camper Singh, Joseph Jaeger, Maya Nyayapati, and Igors Stepanovs. Ratcheted encryption and key exchange: The security of messaging. In *Advances in Cryptology – CRYPTO 2017*, pages 619–650. Springer International Publishing, 2017.
3. Nikita Borisov, Ian Goldberg, and Eric 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.

4. Ran Canetti and Hugo Krawczyk. Analysis of key-exchange protocols and their use for building secure channels. In Birgit Pfitzmann, editor, *Advances in Cryptology — EUROCRYPT 2001*, pages 453–474, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg.
5. Katriel Cohn-Gordon, Cas Cremers, Benjamin Dowling, Luke Garratt, and Douglas Stebila. A formal security analysis of the signal messaging protocol. In *2017 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 451–466, April 2017.
6. Katriel Cohn-Gordon, Cas Cremers, and Luke Garratt. On post-compromise security. In *2016 IEEE 29th Computer Security Foundations Symposium (CSF)*, pages 164–178, June 2016.
7. David Derler, Tibor Jager, Daniel Slamanig, and Christoph Striecks. Bloom filter encryption and applications to efficient forward-secret 0-RTT key exchange. In Jesper Buus Nielsen and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2018*, pages 425–455, Cham, 2018. Springer International Publishing.
8. Yevgeniy Dodis, Michael J. Freedman, Stanislaw Jarecki, and Shabsi Walfish. Optimal signcryption from any trapdoor permutation. Available at: `https://eprint.iacr.org/2004/020.pdf`.
9. Felix Günther, Britta Hale, Tibor Jager, and Sebastian Lauer. 0-RTT key exchange with full forward secrecy. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *Advances in Cryptology – EUROCRYPT 2017*, pages 519–548, Cham, 2017. Springer International Publishing.
10. Joseph Jaeger and Igors Stepanovs. Optimal channel security against fine-grained state compromise: The safety of messaging. Available at: `https://eprint.iacr.org/2018/553.pdf`.
11. Daniel Jost, Ueli Maurer, and Marta Mularczyk. Efficient ratcheting: Almost-optimal guarantees for secure messaging. Available at: `https://eprint.iacr.org/2018/954.pdf`.
12. Brian LaMacchia, Kristin Lauter, and Anton Mityagin. Stronger security of authenticated key exchange. In Willy Susilo, Joseph K. Liu, and Yi Mu, editors, *Provable Security*, pages 1–16, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.
13. Miyako Ohkubo, Koutarou Suzuki, and Shingo Kinoshita. Cryptographic approach to "privacy-friendly" tags. In *RFID Privacy Workshop*, 2003.
14. Miyako Ohkubo, Koutarou Suzuki, and Shingo Kinoshita. Efficient hash-chain based RFID privacy protection scheme. In *International Conference on Ubiquitous Computing (Ubicomp), Workshop Privacy: Current Status and Future Directions*, 2004.
15. Bertram Poettering and Paul Rösler. Ratcheted key exchange, revisited. Available at: `https://eprint.iacr.org/2018/296.pdf`.
16. Open Whisper Systems. Signal protocol library for Java/Android. GitHub repository `https://github.com/WhisperSystems/libsignal-protocol-java`, 2017.
17. Nik Unger, Sergej Dechand, Joseph Bonneau, Sascha Fahl, Henning Perl, Ian Goldberg, and Matthew Smith. SoK: Secure messaging. In *2015 IEEE Symposium on Security and Privacy*, pages 232–249, May 2015.
18. Serge Vaudenay. Adversarial correctness favors laziness. Presented at the CRYPTO 2018 Rump Session.
19. WhatsApp. Whatsapp encryption overview. Technical white paper, available at: `https://www.whatsapp.com/security/WhatsApp-Security-Whitepaper.pdf`, 2016.

A Used Definitions

Function families and collision-resistant hash functions. A function family H defines an algorithm $\mathsf{H.Gen}(1^\lambda)$ which generates a key hk (we may denote its length as $\mathsf{H.kl}$) and a deterministic algorithm $\mathsf{H.Eval}(\mathsf{hk}, m)$ which takes a key hk and a message m to produce a digest of fixed length (we may denote it by $\mathsf{H.ln}$). We will need a collision-resistant hash function H. It should be intractable, given a honestly generated hashing key hk, to find two different messages m and m' such that $\mathsf{H.Eval}(\mathsf{hk}, m) = \mathsf{H.Eval}(\mathsf{hk}, m')$.

Definition 25 (Collision-resistant hash function). *We say that a function family* H *is* $(\mathsf{T}, \varepsilon)$*-collision resistant if for any adversary* $\mathcal{A}$ *limited to time complexity* T*, the probability to win is bounded by* ε*.*

1: $\mathsf{H.Gen}(1^\lambda) \xrightarrow{\$} \mathsf{hk}$
2: $\mathcal{A}(\mathsf{hk}) \xrightarrow{\$} (m_1, m_2)$
3: ***if*** $\mathsf{H.Eval}(\mathsf{hk}, m_1) = \mathsf{H.Eval}(\mathsf{hk}, m_2)$ *and* $m_1 \neq m_2$ ***then*** *win*

Signcryption. Our construction is based on signcryption. Actually, we do not use a strong signcryption scheme as defined by Dodis et al. [8] but rather a naive combination of signature and encryption. We only want that it encrypts and authenticates at the same time. We take the following definition for our naive signcryption scheme.

Definition 26 (Signcryption scheme). *A* signcryption scheme SC *consists of four algorithms: two key generation algorithms* $\mathsf{Gen_S}(1^\lambda) \xrightarrow{\$} (\mathsf{sk_S}, \mathsf{pk_S})$*; and* $\mathsf{Gen_R}(1^\lambda) \xrightarrow{\$} (\mathsf{sk_R}, \mathsf{pk_R})$*; an encryption algorithm* $\mathsf{Enc}(\mathsf{sk_S}, \mathsf{pk_R}, \mathsf{ad}, \mathsf{pt}) \xrightarrow{\$} \mathsf{ct}$*; a decryption algorithm* $\mathsf{Dec}(\mathsf{sk_R}, \mathsf{pk_S}, \mathsf{ad}, \mathsf{ct}) \rightarrow \mathsf{pt}$ *returning a plaintext or* $\perp$*. The correctness property is that for all* pt *and* ad*,*

$$\Pr[\mathsf{Dec}(\mathsf{sk_R}, \mathsf{pk_S}, \mathsf{ad}, \mathsf{Enc}(\mathsf{sk_S}, \mathsf{pk_R}, \mathsf{ad}, \mathsf{pt})) = \mathsf{pt}] = 1$$

when the keys are generated with Gen*.*

This notion comes with two security notions.

Definition 27 (EF-OTCPA). *A signcryption scheme* $(\mathsf{T}, \varepsilon)$*-resists to* existential forgeries under one-time chosen plaintext attacks (EF-OTCPA) *if for any adversary* $\mathcal{A}$ *limited to time complexity* T *playing the following game, the probability to win is bounded by* ε*.*

1: $\mathsf{Gen_S}(1^\lambda) \xrightarrow{\$} (\mathsf{sk_S}, \mathsf{pk_S})$
2: $\mathsf{Gen_R}(1^\lambda) \xrightarrow{\$} (\mathsf{sk_R}, \mathsf{pk_R})$
3: $\mathcal{A}(\mathsf{sk_R}, \mathsf{pk_S}, \mathsf{pk_R}) \xrightarrow{\$} (\mathsf{st}, \mathsf{ad}, \mathsf{pt})$
4: $\mathsf{Enc}(\mathsf{sk_S}, \mathsf{pk_R}, \mathsf{ad}, \mathsf{pt}) \xrightarrow{\$} \mathsf{ct}$
5: $\mathcal{A}(\mathsf{st}, \mathsf{ct}) \xrightarrow{\$} (\mathsf{ad}', \mathsf{ct}')$
6: ***if*** $(\mathsf{ad}, \mathsf{ct}) = (\mathsf{ad}', \mathsf{ct}')$ ***then*** *abort*
7: $\mathsf{Dec}(\mathsf{sk_R}, \mathsf{pk_S}, \mathsf{ad}', \mathsf{ct}') \rightarrow \mathsf{pt}'$
8: ***if*** $\mathsf{pt}' = \perp$ ***then*** *abort*
9: *the adversary wins*

Definition 28 (IND-CCA). *A signcryption scheme is* $(\mathsf{q}, \mathsf{T}, \varepsilon)$-IND-CCA-secure *if for any adversary* $\mathcal{A}$ *limited to* q *queries and time complexity* T*, playing the following game, the advantage* $\Pr[\mathsf{IND\text{-}CCA}_0^{\mathcal{A}} \xrightarrow{\$} 1] - \Pr[\mathsf{IND\text{-}CCA}_1^{\mathcal{A}} \xrightarrow{\$} 1]$ *is bounded by* ε*.*

Game $\mathsf{IND\text{-}CCA}_b^{\mathcal{A}}$

$\mathsf{challenge} = \perp$
$\mathsf{Gen}_S(1^\lambda) \xrightarrow{\$} (\mathsf{sk}_S, \mathsf{pk}_S)$
$\mathsf{Gen}_R(1^\lambda) \xrightarrow{\$} (\mathsf{sk}_R, \mathsf{pk}_R)$
$\mathcal{A}^{\mathsf{Ch},\mathsf{Dec}}(\mathsf{sk}_S, \mathsf{pk}_S, \mathsf{pk}_R) \xrightarrow{\$} b'$
return b'

Oracle $\mathsf{Dec}(\mathsf{ad}, \mathsf{ct})$

if $(\mathsf{ad}, \mathsf{ct}) = \mathsf{challenge}$ ***then*** *abort*
$\mathsf{Dec}(\mathsf{sk}_R, \mathsf{pk}_S, \mathsf{ad}, \mathsf{ct}) \rightarrow \mathsf{pt}$
return pt

Oracle $\mathsf{Ch}(\mathsf{ad}, \mathsf{pt})$

if $\mathsf{challenge} \neq \perp$ ***then*** *abort*
if $b = 0$ ***then*** *replace* pt *by a random message of same length*
$\mathsf{Enc}(\mathsf{sk}_S, \mathsf{pk}_R, \mathsf{ad}, \mathsf{pt}) \xrightarrow{\$} \mathsf{ct}$
$\mathsf{challenge} \leftarrow (\mathsf{ad}, \mathsf{ct})$
return ct

Clearly, we can work with the naive signcryption scheme defined by

$$\mathsf{SC.Enc}(\mathsf{sk}_S, \mathsf{pk}_R, \mathsf{ad}, \mathsf{pt}) = \mathsf{PKC.Enc}(\mathsf{pk}_R, (\mathsf{pt}, \mathsf{DSS.Sign}(\mathsf{sk}_S, (\mathsf{ad}, \mathsf{pt}))))$$

using an IND-CCA-secure public-key cryptosystem PKC and a EF-OTCMA-secure digital signature scheme DSS.

B $\mathsf{C}_{\mathsf{forge}}^{\mathsf{P}_{\mathsf{test}}}$ Forbids More Than Necessary

Let us consider $\mathsf{SC.Enc}(\mathsf{sk}_S, \mathsf{pk}_R, \mathsf{pt}) = \mathsf{PKC.Enc}(\mathsf{pk}_R, \mathsf{pt})$ (which does not use $\mathsf{sk}_S/\mathsf{pk}_S$), where PKC is an IND-CCA-secure cryptosystem without the plaintext aware (PA) security. Hence, there exists an algorithm $C(\mathsf{pk}_R; r) = \mathsf{ct}$ such that $(\mathsf{pk}_R, r, \mathsf{PKC.Dec}(\mathsf{sk}_R, \mathsf{ct}))$ and $(\mathsf{pk}_R, r, \mathsf{random})$ are indistinguishable.[4] We can show that the uniARK obtained from the uniARCAD of Fig. 6 has $(\mathsf{C}_{\mathsf{leak}} \wedge \mathsf{C}_{\mathsf{forge}}^{\mathsf{P}_{\mathsf{test}}})$-KIND security. We can consider the following adversary:

$\mathsf{EXP}_{\mathsf{st}}(S) \rightarrow \mathsf{pk}_R$
pick r; $C(\mathsf{pk}_R; r) \rightarrow \mathsf{ct}$
$\mathsf{RATCH}(R, \mathsf{rec}, \mathsf{ct}) \rightarrow \mathsf{true}$
$\mathsf{TEST}(R) \rightarrow K^*$

Due to the non-PA security, we do have privacy for the tested key. However, this adversary is ruled out by $\mathsf{C}_{\mathsf{forge}}^{\mathsf{P}_{\mathsf{test}}}$. Hence, this cleanness predicate does forbid more than necessary: we have KIND security for more attacks than allowed.

C Comparison with Bellare et al. [2]

Bellare et al. [2] consider uniARK. They consider the KIND security defined by the game on Fig. 7 (with slightly adapted notations). This game has a single exposure oracle revealing the state st, the key k, and also the last used coins, but for the sender only. It also allows multiple TEST queries.

[4] As an example, we can start from an IND-CCA-secure PKC_0 and add a ciphertext in the public key to define PKC. PKC.Gen: $\mathsf{PKC}_0.\mathsf{Gen} \rightarrow (\mathsf{sk}, \mathsf{pk}_0)$; pick x; $\mathsf{PKC}_0.\mathsf{Enc}(\mathsf{pk}, x) \rightarrow y$; $\mathsf{pk} \leftarrow (\mathsf{pk}_0, y)$. Set Enc and Dec the same in PKC_0 and PKC. Then $C(\mathsf{pk}; r) = y$. PKC is also IND-CCA-secure and C has the required property.

In the KIND game, the restricted flag is set when there is a trivial forgery. (It could be unset by receiving a genuine upd but we can ignore it for schemes with RECOVER security.) We can easily see that the cleanness notion required by the TEST queries corresponds to $\mathsf{C}_{\mathsf{leak}} \wedge \mathsf{C}^{\mathsf{P}_{\mathsf{test}}}_{\mathsf{trivial\ forge}} \wedge \mathsf{C}_{\mathsf{noEXP}(\mathsf{R})}$.

Game $\mathsf{KIND}^{\mathcal{A}}_{b}$

$i_s \leftarrow 0;\ i_r \leftarrow 0$

$\mathsf{Init}(1^\lambda) \xrightarrow{\$} (\mathsf{st}_S, \mathsf{st}_R, z)$

pick k

$k_s \leftarrow k\ ;\ k_R \leftarrow k$

$b' \xleftarrow{\$} \mathcal{A}^{\mathsf{RATSEND},\mathsf{RATREC},\mathsf{EXP},\mathsf{CHSEND},\mathsf{CHREC}}(z)$

return b'

Oracle EXP

if $\mathsf{op}[i_s] =$ "ch" **then return** $\perp$

$\mathsf{op}[i_s] =$ "exp"

return (r, st_S, k_S)

Oracle RATSEND

pick r; $(\mathsf{st}'_S, \mathsf{upd}_S, k_S) \leftarrow \mathsf{Send}(\mathsf{st}_S; r)$

$\mathsf{auth}[i_s] \leftarrow \mathsf{upd}$; $i_s \leftarrow i_s + 1$

return upd

Oracle RATREC(upd)

$(\mathsf{acc}, \mathsf{st}_R, k_R) \leftarrow \mathsf{Receive}(\mathsf{st}_R, \mathsf{upd})$

if not acc **then return** false

if $\mathsf{op}[i_r] =$ "exp" **then** restricted $\leftarrow$ true

if $\mathsf{upd} = \mathsf{auth}[i_r]$ **then** restricted $\leftarrow$ false

$i_r \leftarrow i_r + 1$; **return** true

Oracle CHSEND

if $\mathsf{op}[i_s] =$ "exp" **then return** $\perp$

$\mathsf{op}[i_s] \leftarrow$ "ch"

if $\mathsf{rkey}[i_s] = \perp$ **then** $\mathsf{rkey}[i_s] \xleftarrow{\$} \{0,1\}^{\mathsf{kl}}$

if $b = 1$ **then return** k_s **else return** $\mathsf{rkey}[i_s]$

Oracle CHREC

if restricted **then return** k_R

if $\mathsf{op}[i_r] =$ "exp" **then return** $\perp$

$\mathsf{op}[i_r] \leftarrow$ "ch"

if $\mathsf{rkey}[i_r] = \perp$ **then** $\mathsf{rkey}[i_r] \xleftarrow{\$} \{0,1\}^{\mathsf{kl}}$

if $b = 1$ **then return** k_R **else return** $\mathsf{rkey}[i_r]$

Fig. 7: The security game in Bellare et al. [2].

D Comparison with Poettering-Rösler [15]

Poettering and Rösler [15] have a different way to define correctness. Unfortunately, their definition is not complete as it takes schemes doing nothing as correct [18]. Indeed, the trivial scheme letting all states equal to $\perp$ and doing nothing is correct (and obviously secure).

The Poettering-Rösler construction allows to generate keys while treating "associated data" ad at the same time. However, their security notion does not seem to imply authentication of ad although their proposed protocol does. Like ours, this construction method starts from unidirectional, but their uniARK is not FORGE-secure as the state of the receiver allows to forge messages. Another important difference is that their scheme erases the state of the receiver as soon as the reception of an upd fails, instead of just rejecting it and waiting for a correct one. This makes their scheme vulnerable to denial-of-services attack.

The scheme construction uses no encryption. It also accumulates many keys in states, but instead of using an onion encryption, it does many parallel KEM and combines all generated keys as input to a random oracle. They feed the random oracle with the local history of communication as well (instead of using a collision-resistant hash function). It uses a KEM with a special additional property which could be realized with a hierarchical identity-based encryption (HIBE). Instead, we use a signcryption scheme. Finally, it uses the output of the random oracle to generate a new sk/pk pair. One of the participants erases sk and keeps pk while the other keeps sk. In our construction, one participant generates the pair, sends sk to the other, and erases it.

We recall the KIND game of Poettering-Rösler [15] on Fig. 8 (with slightly adapted notations). The adversary can make several TEST queries. Furthermore, $\mathsf{TEST}(\mathsf{P})$ queries

Game $\mathsf{KIND}_b^{\mathcal{A}}$

for $P \in \{A, B\}$ **do**
  $s_P, r_P \leftarrow 0$
  ▷ number of sent and received messages
  $e_P \leftarrow 0$
  ▷ e_P: number of in-sync received messages
  $\mathsf{EP}_P[\cdot] \leftarrow \bot$ ▷ $\mathsf{EP}_P[s]$: value of e_P at the s^{th} send
  $E_P^{\vdash}, E_P^{\dashv} \leftarrow 0$
  ▷ $E_P^{\vdash}$: number of in-sync sent acked by $\overline{P}$
  ▷ $E_P^{\dashv} \leftarrow 0$: number of in-sync sent messages
  $\mathsf{adc}_P[\cdot] \leftarrow \bot$ ▷ list of sent $(\mathsf{ad}, \mathsf{upd})$
  $\mathsf{is}_P \leftarrow \mathsf{true}$ ▷ is_P says if P is in-sync
  $k_P[\cdot] \leftarrow \bot$, $\mathsf{XP}_P \leftarrow \emptyset$ ▷ list of s during $\mathsf{EXP}_{\mathsf{st}}(P)$
  $\mathsf{TR}_P \leftarrow \emptyset$ ▷ list of forbidden $\mathsf{TEST}(P, \ldots)$
  $\mathsf{CH}_P \leftarrow \emptyset$ ▷ list of $\mathsf{TEST}(P, \ldots)$ made
end for
$\mathsf{Init}(1^\lambda) \xrightarrow{\$} (\mathsf{st}_A, \mathsf{st}_B)$
$b' \leftarrow \mathcal{A}^{\mathsf{RATSEND}, \mathsf{RATREC}, \mathsf{EXPst}, \mathsf{EXTkey}, \mathsf{TEST}}()$
if $\mathsf{TR}_A \cap \mathsf{CH}_A \neq \emptyset$ or $\mathsf{TR}_B \cap \mathsf{CH}_B \neq \emptyset$ **then** abort
if $\mathsf{TR}_B \cap \mathsf{CH}_B \neq \emptyset$ or $\mathsf{TR}_B \cap \mathsf{CH}_B \neq \emptyset$ **then** abort
return b'

Oracle $\mathsf{RATSEND}(P, \mathsf{ad})$

if $S_P = \bot$ **then** abort
$(\mathsf{st}_P, k, \mathsf{upd}) \leftarrow \mathsf{Send}(\mathsf{st}_P, \mathsf{ad})$
if is_P **then**
  $\mathsf{adc}_P[s_P] \leftarrow (\mathsf{ad}, \mathsf{upd})$
  $\mathsf{EP}_P[s_P] \leftarrow e_P$
  $E_P^{\dashv} \leftarrow E_P^{\dashv} + 1$
end if
$k_P[P, e_P, s_P] \leftarrow k$
$s_P \leftarrow s_P + 1$
return upd

Oracle $\mathsf{EXP}_{\mathsf{key}}(P, \mathsf{role}, e, s)$

if $k_P[\mathsf{role}, e, s] \in \{\bot, \diamond\}$ **then** abort ▷ not allowed if k_P is not defined or is available from $k_{\overline{P}}$
$k \leftarrow k_P[\mathsf{role}, e, s]$
$k_P[\mathsf{role}, e, s] \leftarrow \diamond$
return k

Oracle $\mathsf{RATREC}(P, \mathsf{ad}, \mathsf{upd})$

if $S_P = \bot$ **then** abort
if $\mathsf{is}_P \wedge \mathsf{adc}_{\overline{P}}[r_P] \neq (\mathsf{ad}, \mathsf{upd})$ **then** ▷ first forgery
  $\mathsf{is}_P \leftarrow \mathsf{false}$
  if $r_P \in \mathsf{XP}_{\overline{P}}$ **then** ▷ trivial forgery
    $\mathsf{TR}_P \leftarrow \mathsf{TR}_P \cup \{\mathsf{send}\} \times \{0, 1, \ldots\} \times \{s_P, s_P + 1, \ldots\}$
    $\mathsf{TR}_P \leftarrow \mathsf{TR}_P \cup \{\mathsf{rec}\} \times \{0, 1, \ldots\} \times \{r_P, r_P + 1, \ldots\}$
  end if
end if
if is_P **then**
  $E_P^{\vdash} \leftarrow \mathsf{EP}_{\overline{P}}[r_P]$
  $e_P \leftarrow e_P + 1$
end if
$(\mathsf{st}_P, k) \leftarrow \mathsf{Receive}(\mathsf{st}_P, \mathsf{ad}, \mathsf{upd})$
if $\mathsf{st}_P = \bot$ **then return** $\bot$
if is_P **then** $k \leftarrow \diamond$ ▷ k is already available on $\overline{P}$
$k_P[\mathsf{rec}, E_P^{\vdash}, r_P] \leftarrow k$
$r_P \leftarrow r_P + 1$
return

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

$\mathsf{TR}_P \leftarrow \mathsf{TR}_P \cup \{\mathsf{rec}\} \times \{E_P^{\vdash}, \ldots, E_P^{\dashv}\} \times \{r_P, r_P + 1, \ldots\}$
if is_P **then**
  $\mathsf{XP}_P \leftarrow \mathsf{XP}_P \cup \{s_P\}$
  $\mathsf{TR}_{\overline{P}} \leftarrow \mathsf{TR}_{\overline{P}} \cup \{\mathsf{send}\} \times \{E_P^{\vdash}, \ldots, E_P^{\dashv}\} \times \{r_P, r_P + 1, \ldots\}$
end if
return st_P

Oracle $\mathsf{TEST}(P, \mathsf{role}, e, s)$

if $k_P[\mathsf{role}, e, s] \in \{\bot, \diamond\}$ **then** abort
$k \leftarrow k_P[\mathsf{role}, e, s]$
if $b = 0$ **then** $k \leftarrow$ random
$k_P[\mathsf{role}, e, s] \leftarrow \diamond$
$\mathsf{CH}_P \leftarrow \mathsf{CH}_P \cup \{(\mathsf{role}, e, s)\}$
return k

Fig. 8: The KIND game of Poettering-Rösler [15].

are not necessarily on the last active k_P but can be on any previously generated k_P value. For this reason, TEST takes as input the index (a triplet (role, e, s)) of the tested key. This does not change the security notion.

The KIND game keeps a flag is_P stating if P is "in-sync". It means that P did not receive any forgery. This is a bit weaker than our matching status. However, assuming that a protocol is such that participants who received a forgery are no longer able to send valid messages to their counterparts, in-sync is equivalent to the matching status. As we can see, a key k_P produced during a reception is erased if P is in-sync, because it is available on the $\overline{P}$ side from where it could be tested. This is one way to rule out some trivial attacks.

The other way is to mark a TEST as forbidden in a TR list. We can see in the KIND game (Step 2–8 in RATREC) that if P receives a trivial forgery (this is deduced by $r_P \in \mathsf{XP}_{\overline{P}}$), then no further $\mathsf{TEST}(P)$ is allowed. This means that $C^{P_{\mathsf{test}}}_{\mathsf{trivial\ forge}}$ is included in the cleanness predicate of this KIND game.

We can easily check that C_{leak} is included in the cleanness predicate. Hence, this KIND game looks equivalent to ours with cleanness predicate $C_{\mathsf{leak}} \wedge C^{P_{\mathsf{test}}}_{\mathsf{trivial\ forge}}$.

This security notion does not seem to imply FORGE security.

E Comparison with Jaeger-Stepanovs [10]

We recall the AEAC game of Jaeger-Stepanovs [10] on Fig. 9 (with slightly adapted notations). The $\mathsf{RATSEND}$ oracle implements the left-or-right challenge at the same time. Hence, the adversary can make several challenges. Additionally, the RATREC oracle implements a decrypt-or-silent oracle which leaks b in the case of a non-trivial forgery. (The oracle always decrypts after a trivial forgery and never decrypts if no forgery. Its behavior changes only in the presence of a non-trivial forgery and with no previous trivial forgery.) Hence, FORGE security is implied by AEAC security. A novelty here is that the adversary can get the *next* random coins to be used: z_P for sending or η_P for receiving. (Bellare et al. [2] allowed to expose the *last* coins.) This is managed by all instructions in gray on Fig. 9. Extracting these coins must be followed by the appropriate oracle query (enforced by the nextop state).

We cannot challenge P after P received a trivial forgery (due to the $\mathsf{restricted}_P$ flag). Hence, we have some kind of $C^{P_{\mathsf{test}}}_{\mathsf{trivial\ forge}}$ condition for cleanness. Since C_{leak} is necessary, we can say that this model includes the $C_{\mathsf{leak}} \wedge C^{P_{\mathsf{test}}}_{\mathsf{trivial\ forge}}$ predicate.

Game $\mathsf{AEAC}_b^{\mathcal{A}}$

for $P \in \{A, B\}$ **do**

  $s_P, r_P \leftarrow 0$

  $\mathsf{restricted}_P \leftarrow \mathsf{false}$ ▷ P received a trivial forgery

  $\mathsf{forge}_P[\cdot] \leftarrow \mathsf{nontrivial}$ ▷ $\mathsf{forge}_P[r]$ says if r^{th} reception could be a trivial forgery

  $\mathcal{X}_P \leftarrow 0$ ▷ challenge forbidden if $r_P < \mathcal{X}_P$ because some $\mathsf{EXP}_{\mathsf{st}}(\overline{P})$ occurred

  pick z_P, η_P

end for

$(\mathsf{st}_A, \mathsf{st}_B) \leftarrow \mathsf{Init}(1^\lambda)$

$b' \leftarrow \mathcal{A}^{\mathsf{RATSEND},\mathsf{RATREC},\mathsf{EXPst}}()$

return b'

Oracle $\mathsf{RATSEND}(P, \mathsf{pt}_0, \mathsf{pt}_1, \mathsf{ad})$

if $\mathsf{nextop} \notin \{(P, \mathsf{send}), \perp\}$ **then return** $\perp$

if $|\mathsf{pt}_0| \neq |\mathsf{pt}_1|$ **then return** $\perp$

if $(r_P < \mathcal{X}_P \vee \mathsf{restricted}_P \vee \mathsf{ch}_P[s_P + 1] = \mathsf{forbidden}) \wedge \mathsf{pt}_0 \neq \mathsf{pt}_1$ **then return** $\perp$

$(\mathsf{st}_P, \mathsf{ct}) \leftarrow \mathsf{Send}(\mathsf{st}_P, \mathsf{ad}, \mathsf{pt}_b; z_P)$

$\mathsf{nextop} \leftarrow \perp$, $s_P \leftarrow s_P + 1$, pick z_P

if $\neg\mathsf{restricted}_P$ **then** $\mathsf{ctable}_{\overline{P}}[s_P] \leftarrow (\mathsf{ct}, \mathsf{ad})$

▷ register ct if P had no trivial forgery

if $\mathsf{pt}_0 \neq \mathsf{pt}_1$ **then** $\mathsf{ch}_P[s_P] \leftarrow \mathsf{done}$

▷ challenge was done for the s^{th} send

return ct

Oracle $\mathsf{RATREC}(P, \mathsf{ct}, \mathsf{ad})$

if $\mathsf{nextop} \notin \{(P, \mathsf{rec}), \perp\}$ **then return** $\perp$

$(\mathsf{st}_P, \mathsf{pt}) \leftarrow \mathsf{Receive}(\mathsf{st}_P, \mathsf{ad}, \mathsf{ct}; \eta_P)$

$\mathsf{nextop} \leftarrow \perp$, pick η_P

if $\mathsf{pt} = \perp$ **then return** $\perp$

$r_P \leftarrow r_P + 1$

if $\mathsf{forge}_P[r_P] = \mathsf{trivial} \wedge (\mathsf{ct}, \mathsf{ad}) \neq \mathsf{ctable}_P[r_P]$ **then** $\mathsf{restricted}_P \leftarrow \mathsf{true}$ ▷ trivial forgery

if $\mathsf{restricted}_P \vee (b = 0 \wedge (\mathsf{ct}, \mathsf{ad}) \neq \mathsf{ctable}_P[r_P])$ **then return** pt ▷ return pt only after trivial forgeries

▷ ($b = 0$ case) return pt for a non-trivial forgery

return $\perp$

Oracle $\mathsf{EXP}_{\mathsf{st}}(P, \mathsf{coins})$

if $\mathsf{nextop} \neq \perp$ **then return** $\perp$

if $\mathsf{restricted}_P$ **then return** $(\mathsf{st}_P, z_P, \eta_P)$

if $\exists i : r_P < i \leqslant s_{\overline{P}} \wedge \mathsf{ch}_{\overline{P}}[i] = \mathsf{done}$ **then return** $\perp$

▷ challenge from $\overline{P}$ was done but not received yet

$\mathsf{forge}_{\overline{P}}[s_P + 1] \leftarrow \mathsf{trivial}$, $z, \eta \leftarrow \perp$, $\mathcal{X}_{\overline{P}} \leftarrow s_P + 1$

if $\mathsf{coins} = \mathsf{send}$ **then**

  $\mathsf{nextop} \leftarrow (P, \mathsf{send})$, $z \leftarrow z_P$, $\mathcal{X}_{\overline{P}} \leftarrow s_P + 2$

  $\mathsf{forge}_{\overline{P}}[s_P + 1] \leftarrow \mathsf{trivial}$, $\mathsf{ch}_P[s_P + 2] \leftarrow \mathsf{forbidden}$

else if $\mathsf{coins} = \mathsf{rec}$ **then**

  $\mathsf{nextop} \leftarrow (P, \mathsf{rec})$, $\eta \leftarrow \eta_P$

end if

return (st_P, z, η)

Fig. 9: The AEAC game of Jaeger-Stepanovs [10].