Symmetric Key Exchange with Full Forward Security and Robust Synchronization

Colin Boyd[1], Gareth T. Davies[2][0000−0002−5935−5725], Bor de Kock[1][0000−0003−3143−4381], Kai Gellert[2][0000−0003−0985−7265], Tibor Jager[2], and Lise Millerjord[1]

[1] NTNU – Norwegian University of Science and Technology, Trondheim, Norway
[2] Bergische Universität Wuppertal, Wuppertal, Germany

Abstract. We construct lightweight authenticated key exchange protocols based on pre-shared keys, which achieve *full* forward security and rely only on simple and efficient symmetric-key primitives. All of our protocols have rigorous security proofs in a strong security model, all have low communication complexity, and are particularly suitable for resource-constrained devices.

We describe three protocols that apply *linear* key evolution to provide different performance and security properties. *Correctness* in parallel and concurrent protocol sessions is difficult to achieve for linearly key-evolving protocols, emphasizing the need for assurance of availability alongside the usual confidentiality and authentication security goals. We introduce *synchronization robustness* as a new formal security goal, which essentially guarantees that parties can re-synchronize efficiently. All of our new protocols achieve this property.

Since protocols based on linear key evolution cannot guarantee that all concurrently initiated sessions successfully derive a key, we also propose two constructions with *non-linear* key evolution based on puncturable PRFs. These are instantiable from standard hash functions and require $O(C \cdot \log(|\mathsf{CTR}|))$ memory, where C is the number of concurrent sessions and $|\mathsf{CTR}|$ is an upper bound on the total number of sessions per party. These are the first protocols to simultaneously achieve full forward security, synchronization robustness, and concurrent correctness.

1 Introduction

Authenticated key exchange protocols based on pre-shared long-term symmetric keys (PSK-AKE) enable two parties to use a previously established symmetric key, agreed upon via out-of-band communication, to (mutually) authenticate and

This work was supported by Deutscher Akademischer Austauschdienst (DAAD) and Norges forskningsråd (NFR) under the PPP-Norwegen programme. Colin Boyd and Lise Millerjord have been supported by NFR project number 288545. Tibor Jager and Gareth T. Davies have been supported by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme, grant agreement 802823.

derive a shared session key. Prominent examples of such protocols are the PSK modes of TLS 1.3 and prior TLS versions, but these examples still make use of public-key techniques for key derivation, even if authentication uses symmetric keys. PSK-AKE protocols can be significantly more efficient than classical public-key AKE protocols, particularly when they can be constructed exclusively based on symmetric key primitives ("symmetric AKE") for both authentication and key derivation. Therefore such protocols are especially desirable for performance-constrained devices, such as battery-powered wireless IoT devices, where every computation and every transmitted bit has a negative impact on battery life. More generally, such protocols may be preferable in "closed-world" applications, such as industrial settings, where pre-sharing keys may be easier and more practical than deploying a public-key infrastructure. Furthermore, protocols based purely on symmetric-key techniques, such as hash functions and symmetric encryption, also achieve security against quantum attacks by adjusting security parameters appropriately.

Forward Security in Symmetric AKE Protocols. Forward security is today a standard security goal of key exchange protocols. It requires that past session keys remain secure, even if the secret long-term key material is compromised. Note that this is only achievable if past session keys are *not* efficiently computable from a current long-term key. Forward security is comparatively easily achievable if *public key* cryptography is used. For instance, a classical approach is to use *ephemeral* keys for key establishment, such as the Diffie-Hellman protocol or, more generally, a key encapsulation mechanism (KEM). *Independent* long-term keys can then be used for authentication via digital signatures or another KEM.

The only currently known way to avoid public key techniques and use only symmetric key primitives is based on the *"derive-then-evolve"* approach, where first a session key is derived from a long-term key, and then the long-term key is evolved. This key evolution prevents efficient re-computation of prior session keys which yields forward security. Both steps can be implemented with simple key derivation functions. There are two common ways to use this approach:

1. *Synchronized key evolution.* In this case, both parties evolve their long-term keys in "epochs", e.g., once per day. Note that this approach cannot achieve *"full"* foward security, but only a weaker *"delayed"* form. This is because all session keys of the current epoch can be computed from the current long-term secret, so forward security only holds for session keys of past epochs. Moreover, this approach requires synchronized clocks between parties, even to achieve correctness. For many applications this seems impractical, in particular for cheap low-performance devices, for which symmetric AKE protocols are particularly relevant.
2. *Triggered key evolution.* In this case the protocol ensures that both parties advance their key material during protocol execution. This approach directly achieves "full" forward security for every session, and therefore seems preferable. However, this apparently simple approach turns out to be much less trivial to realize than might be expected, because both parties must remain

"in sync", such that correctness is guaranteed even in presence of *concurrent* sessions or message loss due to network failures or *active attacks*. This approach has similarities with *ratcheting* [1], but there are significant differences in our setting as discussed under *Related Work* below.

Concurrency and Key-Evolving Protocols. The possibility of running concurrent protocol sessions in parallel is a standard correctness requirement for protocols, and reflected in all common AKE security models, such as the BR and CK models [8, 15] and their countless variants and refinements. The main technical challenge of key evolution is to achieve full forward security while maintaining *correctness* in the presence of parallel and concurrent protocol sessions.

Even if we assume that all parties are *honest* and that all messages are transmitted *reliably* (i.e., without being dropped because of an unreliable network or influence from an adversary) this is already highly non-trivial and we do not know of any currently existing forward-secure symmetric AKE protocol which achieves *correctness* and full forward security in such a setting. The difficulty is essentially that one session might advance a key "too early" for another concurrent session to be completed, which breaks correctness. No such difficulty appears in classical forward-secure public key protocols, since long-term keys are usually *static* and different sessions use independent randomness. So it turns out that, somewhat surprisingly, forward security and correctness is more difficult to achieve for symmetric AKE.

To complicate matters even further, note that the assumption of honest parties and reliable message transmission is very strong and may not be realistic for many applications. Therefore we actually want to achieve forward security and "synchronization robustness" in the presence of an adversary which intentionally aims to *break* synchronization, e.g., by adaptively dropping or re-ordering messages. Such an adversary is attacking *availability* properties of the AKE protocol, an important aspect of security usually omitted from key exchange security models. The development of techniques to ensure availability for stateful key exchange is an unsolved foundational problem.

Our Contributions. In this work we develop several new lightweight forward-secure symmetric AKE protocols with different efficiency and correctness properties. Table 1 summarizes the main security and efficiency properties of our new protocols. This includes the first protocols that provably achieve synchronization robustness, a formal availability security notion we introduce, and correctness in the presence of concurrent sessions. More concretely we achieve the following.

Security model. We describe a security model suited to forward-secure symmetric AKE capturing entity authentication (one-sided and mutual), indistinguishability of established keys, and forward security. Our model follows a standard approach for AKE protocols based on the Bellare-Rogaway model [8], adapted to the requirements of symmetric AKE with evolving keys.

Table 1: Overview of our protocols and comparison to SAKE [4]. The number in the protocol name indicates the total number of messages per protocol run, "R only" means that only the responder authenticates its communication partner. The third column considers the communication complexity, where **C** is the number of counter values that are sent, **M** the number of MACs, and **N** the number of nonces. **Sync. Rob.** indicates the achieved level of synchronization robustness, **Bd. Gap** whether the gap between two parties is bounded (for non-concurrent executions), **CC** whether concurrent correctness is achieved, and **FS** whether full forward security is achieved.

Protocol	**Auth.**	**(C, M, N)**	**Sync. Rob.**	**Bd. Gap**	**CC**	**FS**
SAKE (5) [4]	mutual	(0,4,2) + ID	✗	✓	✗	✓
SAKE-AM (4) [4]	mutual	(0,4,2) + ID	✗	✓	✗	✓
LP3	mutual	(3,3,2)	weak	✓	✗	✓
LP2	mutual	(2,2,0)	weak	✗	✗	✓
LP1	R only	(1,1,0)	weak	✗	✗	✓
PP2	mutual	(1,2,0)	full	✓	✓	✓
PP1	R only	(1,1,0)	full	✓	✓	✓

Synchronization robustness. We formalize a new property called *synchronization robustness* (SR), which is trivially achieved for traditional AKE protocols with fixed long-term keys, but turns out to be a crucial feature for key-evolving protocols such as forward-secure symmetric AKE. Essentially, SR captures whether parties in a protocol can efficiently re-synchronize their states in order to complete a successful protocol run. This should even hold if an adversary controls the network and/or some of the parties.

We define two flavours. Both consider an active adversary that may execute arbitrary protocol sessions to manipulate the state of parties, and whose goal is to manipulate the state such that a subsequent protocol execution fails.

In *weak* SR the 'target' protocol session must then be executed without adversarial interaction (similar to the corresponding requirement in Krawczyk's weak forward security [26]). "Full" SR allows the adversary arbitrary queries between messages of the 'target' session, even to parties of the oracles involved in the 'target' session.

Linear key evolving protocols. We define the notion of *linear key evolution*, which makes the classical "derive-then-evolve" approach concrete. We argue that protocols based on linear key evolution can only achieve weak SR and cannot achieve concurrent correctness.

We construct three different protocols (LP1, LP2, LP3, cf. Table 1), all of which achieve weak SR. Most interestingly, LP3 even achieves a "bounded gap" property, which means that no active adversary in control of the network is able to force the state of two parties to differ by more than one key

evolving step, so that a party is always able to catch up quickly, if necessary. For all three protocols we show that in a setting where concurrent runs between two parties are allowed, this number of steps required to catch up is bounded in the number of concurrent runs. To this end, we apply a new approach to precisely analyze the state machine of a protocol. Furthermore, we also show two extremely lightweight protocols LP1 and LP2, which provide one-sided and mutual authentication, respectively, and where the communication complexity is only one (resp. two) MAC and one (resp. two) counter value.

Full SR and concurrent correctness. This leads to the question of whether and how full synchronization robustness and concurrent correctness (CC) can be achieved. We propose the use of puncturable pseudorandom functions (PPRFs) to apply a "non-linear" key evolving strategy, and we construct two protocols PP1 and PP2, which both achieve full SR and CC.
Since PPRFs can be efficiently instantiated from cryptographic hash functions, both protocols are extremely lightweight. PP1 achieves one-sided authentication with a single counter value and a single MAC, PP2 mutual authentication with one counter and two MACs. Furthermore, while repeated puncturing PPRFs may lead to large secret keys [3], we take advantage of the stateful nature of symmetric AKE protocols to instantiate the PPRF such that secret key size is at most *logarithmic* in the number of sessions.

Hence, we offer a versatile catalogue of lightweight and forward-secure symmetric AKE protocols with significantly stronger correctness and security properties. This includes the first protocols to achieve concurrent correctness and full synchronization robustness, or weak SR with bounded gap. Which of these protocols is best for a particular application depends on the nature of the security and functionality requirements. Further, in LP3 the parties exchange nonces: we recognize that in some applications sufficient randomness will not be available and so we prove the protocol secure for any nonce generation procedure, which could be random selection or (stateful) use of a counter.

Related Work. Bellare and Yee [9] analyzed forward security for symmetric-key primitives, specifically pseudo-random generation, message authentication codes and symmetric encryption. They provide constructions using key evolution which are similar to the linear key evolution that we employ, and their protocols use some techniques from key-evolving schemes such as prior work on forward-secure signatures [6]. Their work does not deal with key exchange.

Brier and Peyrin [14] gave a tree-based protocol for key establishment, with the stated aim of improving the DUKPT scheme defined in ANSI X9.24 [2]. The idea in DUKPT is that the client device (payment terminal) is highly constrained in terms of memory, yet needs to derive a unique key per transaction from an original pre-shared key, by applying a PRF (based on Triple-DES) to a counter and the base derivation key. Their work involves formalizing the specific problem faced in the payment terminal setting, and their scheme assumes an incorruptible server: a far weaker security model than the one that we consider. A similar

security assumption was used by Le et al. [27], who presented a protocol for use in the context of Radio Frequency Identification (RFID), where the server keeps two values of the key derivation key to deal with potential synchronization loss.

Li et al. [28] analyzed the pre-shared key ciphersuites of TLS 1.2, using an adaption of the ACCE model of Jager et al. [23]. In this setting, Li et al. presented a formalization of the prior AKE-style models, but where parties could share PSK material with other parties in addition to their long-term key pairs.

Dousti and Jalili [18] presented a key exchange protocol called FORSAKES, which is based on synchronized time-based key evolution. Their protocol requires 3 messages and assumes perfect synchronicity of parties to achieve correctness, and as we have already mentioned their approach can only obtain *delayed forward security*. A discussion of delayed forward security and more generally the various challenges involved in defining forward security was given Boyd and Gellert [12].

The concept of evolving symmetric keys is reminiscent of Signal's double ratchet [1], a well-known example of a symmetric protocol with evolving keys. Signal employs a Diffie-Hellman-ratchet, which adds new key material at every step through multiple Diffie-Hellman exchanges along the way. At every step of this main ratchet a separate linear key evolving ratchet is 'branched off', which is similar to how linear evolution works in our protocols — however, a critical difference is that in our scenario we evolve the key shared across different sessions as opposed to evolving a key within one session as happens in the Signal protocol. It is this difference which leads to the complexity of managing synchronization between sessions which run concurrently. In addition to this difference, which anyway makes Signal unusable for our setting, use of Diffie-Hellman in the Signal ratchet means that there is a vector for quantum attacks, while our protocol is purely based on symmetric primitives.

Another primitive conceptually similar to PPRFs is *puncturable encryption*, which was introduced by Green and Miers in 2015 [20], and has since led to several follow-up constructions of puncturable encryption [16,17,21,30]. However, all those constructions rely on expensive public-key techniques (such as bilinear pairings) and are thus impractical in the context of this work.

Comparison with Avoine et al. [4]. In Table 1 above we have mentioned two protocols named SAKE and SAKE-AM that were presented by Avoine et al. [4] (henceforth ACF20). Their paper was the first to provide key exchange protocols that attain forward security via linear evolution. Their system assumptions are largely the same as ours, with the crucial difference that our models are equipped to capture parallel executions. The security model of ACF20 explicitly disallows concurrent sessions, which not only yields a weak security notion, but also sidesteps the major difficulty of achieving even correctness in the presence of concurrent sessions in key-evolving symmetric-key protocols. Indeed, the protocols from ACF20 completely break down when executed concurrently, allowing an adversary to prevent the parties from computing any session keys in future sessions. We consider this an unrealistic and impractical restriction for many applications. Therefore we introduce the new notion of synchronization robust-

ness, which formally defines the ability of key-evolving protocols to deal with concurrent executions, including in adversarial environments.

We embrace the use of (explicit) counters to acquire linear key evolving protocols that are conceptually simpler and require fewer messages than those provided by ACF20, in a way that additionally provides (weak) synchronization robustness. In any protocol that uses PSK evolution to achieve forward security a party must update the key state after a successful protocol run, and in embedded devices this requires writing to persistent storage. Our protocols require the updating (writing) of one key and one counter per session, while SAKE and SAKE-AM require updating two keys. Since a sequentially evolving key can also be seen as an implicit counter, conceptually the distinction between counters and evolving keys seems to be minor. The storage overhead of our protocols compared to ACF20's protocols is the (usually 8-byte) counter, while the linear key evolving protocols in our paper and ACF20 require storage of two keys (usually 16 or 32 bytes).

We note that ACF20 remarked that the parties could use separate PSKs for concurrent executions, however this solution requires an a priori bound on the number of possible concurrent sessions that could occur and a corresponding multiplication in key storage: none of our protocols require this. Further, implementing their approach would require a modification of their protocols, since parties need to know which PSK to use, and the security of these modified protocols is not proven.

Preliminaries. We denote the security parameter as λ. For any $n \in \mathbb{N}$ let 1^n be the unary representation of n and let $[n] = \{1, \ldots, n\}$ be the set of numbers between 1 and n. We write $x \xleftarrow{\$} \mathcal{S}$ to indicate that we choose element x uniformly at random from set $\mathcal{S}$. For a probabilistic polynomial-time algorithm $\mathcal{A}$ we define $y \xleftarrow{\$} \mathcal{A}(a_1, \ldots, a_n)$ as the execution of $\mathcal{A}$ (with fresh random coins) on input $a_1, \ldots, a_n$ and assigning the output to y. The function $\texttt{NextOdd}(x)$ takes as input an integer and ouputs the next odd integer greater than x, i.e. whichever element of $\{x+1, x+2\}$ is odd. Our protocols require the use of counters, and integer $|\mathsf{CTR}|$ is the largest possible counter value. Furthermore, we write $[n] \times [n] \setminus (i^*, j^*)$ as a shorthand for $\{(i,j) \in [n]^2\} \setminus \{(i^*, j^*)$ with $i < j\}$.

1.1 Message Authentication Codes

Throughout this paper we assume that all MACs are deterministic. This is to reduce complexity in our proofs, however most MACs used in practice are deterministic [22, 25].

Definition 1 (Message Authentication Codes). *A* message authentication code *consists of three probabilistic polynomial-time algorithms* $\mathsf{MAC} = (\mathsf{KGen}, \mathsf{Mac}, \mathsf{Vrfy})$ *with key space* $\mathcal{K}_{\mathsf{MAC}}$ *and the following properties:*

- $\mathsf{KGen}(1^\lambda)$ *takes as input a security parameter* λ *and outputs a symmetric key* $\mathsf{K}^{\mathsf{MAC}} \in \mathcal{K}_{\mathsf{MAC}}$*;*

$\mathsf{G}^{\mathsf{SEUF\text{-}CMA}\text{-}Q}_{\mathsf{MAC}}(\mathcal{A})$

1 : $\mathtt{K}^{\mathsf{MAC}} \xleftarrow{\$} \mathsf{KGen}(1^\lambda)$
2 : $\mathcal{Q}, \mathcal{V} \leftarrow \emptyset$
3 : $\mathcal{A}^{\mathcal{O}_{\mathsf{Mac}}(\cdot), \mathcal{O}_{\mathsf{Vrfy}}(\cdot,\cdot)}(1^\lambda)$
4 : **if** $\exists (m, \sigma) \in \mathcal{V} \setminus \mathcal{Q}$
5 : **return** 1
6 : **return** 0

$\mathcal{O}_{\mathsf{Mac}}(m)$

7 : $\sigma \leftarrow \mathsf{Mac}(\mathtt{K}^{\mathsf{MAC}}, m)$
8 : $\mathcal{Q} \coloneqq \mathcal{Q} \cup \{(m, \sigma)\}$
9 : **return** σ

$\mathcal{O}_{\mathsf{Vrfy}}(m, \sigma)$

10 : $b \leftarrow \mathsf{Vrfy}(m, \sigma)$
11 : **if** $b = 1$
12 : $\mathcal{V} \coloneqq \mathcal{V} \cup \{(m, \sigma)\}$
13 : **return** b

Fig. 1: The SEUF-CMA-Q security experiment for message authentication code MAC. $\mathcal{A}$ can make Q queries to $\mathcal{O}_{\mathsf{Vrfy}}$.

- $\mathsf{Mac}(\mathtt{K}^{\mathsf{MAC}}, m)$ *takes as input a key* $\mathtt{K}^{\mathsf{MAC}} \in \mathcal{K}_{\mathsf{MAC}}$ *and a message* m*. Output is a tag* σ*;*
- $\mathsf{Vrfy}(\mathtt{K}^{\mathsf{MAC}}, m, \sigma)$ *takes as input a key* $\mathtt{K}^{\mathsf{MAC}} \in \mathcal{K}_{\mathsf{MAC}}$*, a message* m*, and a tag* σ*. Output is a bit* $b \in \{0, 1\}$*.*

We call a message authentication code correct *if for all* m*, we have*

$$\Pr_{\mathtt{K}^{\mathsf{MAC}} \xleftarrow{\$} \mathsf{KGen}(1^\lambda)} \left[\mathsf{Vrfy}(\mathtt{K}^{\mathsf{MAC}}, m, \mathsf{Mac}(\mathtt{K}^{\mathsf{MAC}}, m)) = 1\right] = 1.$$

We define MAC security as strong existential unforgeability under chosen message attack, where the adversary has access to a verification oracle. In the more common version of this game, which we denote SEUF-CMA-1, the adversary must stop running after it submits its first verification query: this is a subcase of our more general definition. Bellare et al. [5] showed that in the strong unforgeability case these definitions are equivalent up to a loss factor Q.

Definition 2 (MAC Security). *The advantage of an adversary* $\mathcal{A}$ *in the* SEUF-CMA-Q *security experiment defined in Fig. 1 for message authentication code* MAC *is*

$$\mathsf{Adv}^{\mathsf{SEUF\text{-}CMA}\text{-}Q}_{\mathsf{MAC}}(\mathcal{A}) := \Pr\left[\mathsf{G}^{\mathsf{SEUF\text{-}CMA}\text{-}Q}_{\mathsf{MAC}}(\mathcal{A}) = 1\right].$$

1.2 Pseudorandom Functions

Definition 3 (Pseudorandom Functions). *A* pseudrandom function *is a deterministic function* $y = \mathsf{PRF}(k, x)$ *that takes as input some key* $k \in \mathcal{K}_{\mathsf{PRF}}$ *and some element of a domain* $\mathcal{D}_{\mathsf{PRF}}$*, and returns an element* $y \in \mathcal{R}_{\mathsf{PRF}}$*.*

Definition 4 (PRF Security). *The advantage of an adversary* $\mathcal{A}$ *in the* PRF-sec *security experiment defined in Fig. 2 for pseudorandom function* PRF *is*

$$\mathsf{Adv}^{\mathsf{PRF\text{-}sec}}_{\mathsf{PRF}}(\mathcal{A}) := \left|\Pr\left[\mathsf{G}^{\mathsf{PRF\text{-}sec}}_{\mathsf{PRF}}(\mathcal{A}) = 1\right] - \frac{1}{2}\right|.$$

$\mathsf{G}^{\mathsf{PRF\text{-}sec}}_{\mathsf{PRF}}(\mathcal{A})$

$b \xleftarrow{\$} \{0,1\}$
$k_{\mathsf{PRF}} \xleftarrow{\$} \mathcal{K}_{\mathsf{PRF}}$
$g \xleftarrow{\$} \{\mathcal{F} : \mathcal{D}_{\mathsf{PRF}} \to \mathcal{R}_{\mathsf{PRF}}\}$
$b^* \xleftarrow{\$} \mathcal{A}^{\mathcal{O}_f(\cdot)}(1^\lambda)$
if $b^* = b$
  return 1
return 0

$\mathcal{O}_f(x)$

if $b = 1$
  $y \leftarrow f(k_{\mathsf{PRF}}, x)$
else
  $y \leftarrow g(x)$
return y

Fig. 2: The PRF-sec security experiment for pseudorandom function PRF. $\{\mathcal{F} : \mathcal{D}_{\mathsf{PRF}} \to \mathcal{R}_{\mathsf{PRF}}\}$ is the set of all functions from $\mathcal{D}_{\mathsf{PRF}}$ to $\mathcal{R}_{\mathsf{PRF}}$.

2 Authenticated Key Exchange in the Symmetric Setting

In this section we describe our model for authenticated key exchange with forward security in the symmetric setting. Our model follows the standard approach of AKE protocols based on the Bellare-Rogaway model [8], adapted to the requirements of symmetric AKE with evolving keys. This includes definitions for entity authentication (one-sided or mutual), key indistinguishability, and forward security. Furthermore, we define the property of synchronization robustness, which is a crucial feature for forward-secure symmetric key exchange protocols. Parts of our formalization take inspiration from the models of Jager et al. [23].

Differences to public-key AKE models. The most notable difference in the symmetric key setting is that each pair of parties is initialized with shared key material, which is specified before the actual protocol is run. This key material typically includes MAC keys or key derivation keys that have been established in an out-of-band communication (e.g., chosen during the manufacturing process of devices). In order to achieve forward-security via "key evolving techniques" in the symmetric key setting, we additionally have to provide (sessions of) parties with the ability to modify the party's key material. As a consequence, the shared key material of two parties will not always be equal: While one party might evolve their key before preparing the first protocol message, the responder can (at the earliest) evolve after it has received that message.

2.1 Execution Environment

We consider a set of n parties $\{P_1, \ldots, P_n\}$, where each party is a potential protocol participant. We refer to parties by P_i or by their label i if context is clear. Initially, each pair of parties (P_i, P_j) with $i \neq j$ share a common secret $\mathsf{PSK}_{i,j}$, which is the initial key material generated during protocol initialization

(e.g., MAC keys or key derivation keys). Note that this key material may evolve over time and that $\mathsf{PSK}_{i,j}$ and $\mathsf{PSK}_{j,i}$ may not necessarily be equal at all times.

We model parallel executions of a protocol by equipping each party i with $q \in \mathbb{N}$ session oracles $\pi_i^1, \ldots, \pi_i^q$. Each session oracle represents a process that executes one single instance of the protocol. All oracles have access to the "global key material" PSK (including the ability to modify the key material PSK). Moreover, each oracle maintains an internal state consisting of the following variables:

Variable	Description
α	execution state $\in \{\texttt{uninitialized}, \texttt{negotiating}, \texttt{accept}, \texttt{reject}\}$
pid	identity of the intended partner $\in \{P_1, \ldots, P_n\}$
ρ	role $\in \{\mathsf{Initiator}, \mathsf{Responder}\}$
sk	session key $\in \mathcal{K}_\mathsf{s} \cup \perp$ for some session key space $\mathcal{K}_\mathsf{s}$
κ	freshness of session key $\in \{\mathsf{exposed}, \mathsf{fresh}\}$
sid	session identifier
b	security bit $\in \{0, 1\}$

Additionally, we assume that each oracle has an additional temporary state variable, used to store ephemeral values or the transcript of messages. As initial state of the oracle, we have $\alpha = \texttt{uninitialized}$ and $\kappa = \mathsf{fresh}$ and $b \xleftarrow{\$} \{0, 1\}$. Note that pid and ρ are set when the adversary interacts with the respective oracles and that sid and sk are defined as the protocol/adversary progresses.

As usual, if an oracle derives a session key then it will enter the execution state `accept`. If an oracle reaches the execution state `reject`, then it will no longer accept any messages. Later on when we describe protocols, the event `Abort` will identify points at which this action would be triggered.

To begin any of the experiments in this section, the challenger initializes n parties $\{P_1, \ldots, P_n\}$, with each pair of parties sharing symmetric key material PSK as specified by the protocol.

An adversary interacts with session oracles π_i^s by issuing the following queries. Several of these queries add output to an oracle transcript (defined below) which is available to the adversary.

- $\mathsf{NewSessionI}(\pi_i^s, \mathsf{pid})$ initializes a new initiator session for party P_i with intended partner pid. Specifically, this query assigns pid, $\rho = \mathsf{Initiator}$ and $\alpha = \texttt{negotiating}$ to π_i^s, creates the first protocol message and adds this to transcript of π_i^s.
- $\mathsf{NewSessionR}(\pi_i^s, \mathsf{pid}, m)$ initializes a new responder session for party P_i with $\rho = \mathsf{Responder}$ and intended partner pid, and delivers a protocol message to this oracle. Specifically, it assigns pid and $\rho = \mathsf{Responder}$ to π_i^s and processes message m. The message m and consequent protocol messages (if any) are added to its transcript, and the execution state is set to `negotiating`.
- $\mathsf{Send}(\pi_i^s, m)$ delivers message m to oracle π_i^s. This input message, and consequent protocol messages (if any), are added to this oracle's transcript.
- $\mathsf{RevealKey}(\pi_i^s)$ reveals session key sk_i^s and sets $\pi_i^s.\kappa$ to $\mathsf{exposed}$.
- $\mathsf{Corrupt}(P_i, P_j)$ (issued to some oracle of P_i or P_j) returns $\mathsf{PSK}_{i,j}$. If the query $\mathsf{Corrupt}(P_i, P_j)$ is the τ-th query issued by $\mathcal{A}$, we say that all oracles π_i with

$\mathsf{pid} = j$ are τ-corrupted. (i.e., party P_i becomes τ-corrupted with respect to the other party P_j). An uncorrupted oracle is considered as $+\infty$-corrupted.
- Test (π_i^s) chooses $sk_0 \xleftarrow{\$} \mathcal{K}_\mathsf{s}$, sets $sk_1 = \pi_i^s.sk$ and returns sk_b. This oracle can only be queried once, and the query making this action is labelled τ_0.

The adversary must call $\mathsf{NewSessionI}$ or $\mathsf{NewSessionR}$ in order to specify a role and intended partner identifier for each oracle it wishes to use. Afterwards, the adversary can use the Send query to convey messages to these oracles.

2.2 AKE Security

To define entity authentication we use *matching conversations* [8] for oracle partnering, which requires a definition of an oracle's *transcript*: T_i^s is the sequence of all messages sent and received by π_i^s in chronological order. The standard definition of matching conversations, reflects that the party that sends the last message cannot be sure that the responder received that protocol message. We use this definition for entity authentication.

Note that an oracle π_i^s only has a transcript, T_i^s, if $\pi_i^s.\alpha \neq \texttt{uninitialized}$. Transcript T_j^t is a *prefix* of T_i^s if T_j^t contains at least one message and messages in T_j^t are identical to and in the same order as the first $|\mathsf{T}_j^t|$ messages of T_i^s.

Definition 5 (Partial-transcript Matching conversations [23, Def. 3]). *π_i^s has a partial-transcript matching conversation to π_j^t if*

- *T_j^t is a prefix of T_i^s and π_i^s has sent the last message(s), or*
- *$\mathsf{T}_i^s = \mathsf{T}_j^t$ and π_j^t has sent the last message(s).*

However, standard matching conversations are not strong enough to define key indistinguishability in a symmetric setting and leave room for a trivial attack (intuitively, this is due to the "asynchronous evolution" of the global key material PSK). Consider an adversary that uses the above execution environment to execute some protocol between two (sessions of two) parties. The adversary forwards all messages but the last one between both parties. At this point the party that sent the last message must have reached the accept state and applied some one-way procedure to its key material PSK in order to achieve forward security. However, the other party still needs to receive the final message in order to derive the session key and update its version of the key material. If the adversary were now to use Test on the accepting party while using $\mathsf{Corrupt}$ on the other party, this leads to a trivial distinguishing attack in standard key indistinguishability games (e.g., in [23]). Hence, we need to introduce a slightly stronger notion of matching conversations to precisely capture when $\mathsf{Corrupt}$ queries are allowed: the conversation is only deemed to be matching if all messages were delivered.

Definition 6 (Guaranteed Delivery Matching conversations). *π_i^s has a guaranteed delivery matching conversation to π_j^t if $\mathsf{T}_i^s = \mathsf{T}_j^t$.*

As usual, we say that the adversary breaks entity authentication if it forces a fresh oracle to accept maliciously, and breaks key indistinguishability if it can distinguish from random an established key that it cannot trivially access.

Definition 7 (Entity Authentication). *Let Π be a protocol. Let $\mathsf{G}_{\Pi}^{\mathsf{Ent\text{-}Auth}}(\mathcal{A})$ be the following game:*

- *The challenger initializes n parties and their keys;*
- *$\mathcal{A}$ may issue queries to oracles* NewSessionI*,* NewSessionR*,* Send*,* RevealKey*,* Corrupt *and* Test *as defined above;*
- *Once $\mathcal{A}$ has concluded, the experiment outputs 1 if and only if there exists an accepting oracle π_i^s such that the following conditions hold:*
  1. *both P_i (w.r.t. P_j) and intended partner P_j (w.r.t. P_i) were not corrupted before query τ_0;*
  2. *there is no unique π_j^t, with $\rho_i^s \neq \rho_j^t$, such that π_i^s has a partial-transcript matching conversation to π_j^t.*

Define the advantage of an adversary $\mathcal{A}$ in the Ent-Auth *security experiment $\mathsf{G}_{\Pi}^{\mathsf{Ent\text{-}Auth}}(\mathcal{A})$ as*

$$\mathsf{Adv}_{\Pi}^{\mathsf{Ent\text{-}Auth}}(\mathcal{A}) := \Pr\left[\mathsf{G}_{\Pi}^{\mathsf{Ent\text{-}Auth}}(\mathcal{A}) = 1\right].$$

An oracle π_i^s accepting in the above sense 'accepts maliciously'.

Later on we separate the analysis of an initiator oracle accepting maliciously from a responder oracle accepting maliciously. Further, we will present protocols that only provide one-sided authentication: this requires separation of the AKE definition. To this end, we use the following notation:

$$\mathsf{Adv}_{\Pi}^{\mathsf{Ent\text{-}Auth}}(\mathcal{A}) = \mathsf{Adv}_{\Pi}^{\mathsf{Ent\text{-}Auth\text{-}I}}(\mathcal{A}) + \mathsf{Adv}_{\Pi}^{\mathsf{Ent\text{-}Auth\text{-}R}}(\mathcal{A}).$$

Definition 8 (Key Indistinguishability). *Let Π be a protocol. Let $\mathsf{G}_{\Pi}^{\mathsf{Key\text{-}Ind}}(\mathcal{A})$ be the following game:*

- *The challenger initializes n parties and their keys;*
- *$\mathcal{A}$ may issue queries to oracles* NewSessionI*,* NewSessionR*,* Send*,* RevealKey*,* Corrupt *and* Test *as defined above;*
- *Once $\mathcal{A}$ has output (i, s, b') to indicate its conclusion, the experiment outputs 1 if and only if there exists an oracle π_i^s such that the following holds:*
  1. *π_i^s accepts, with a unique oracle π_j^t, such that π_i^s has a partial-transcript matching conversation to π_j^t, when $\mathcal{A}$ issues its τ_0-th query;*
  2. *$\mathcal{A}$ did not issue* RevealKey *to π_i^s nor π_j^t (so $\kappa_i^s = \mathsf{fresh}$) and $\rho_i^s \neq \rho_j^t$;*
  3. *P_i (w.r.t. P_j) is τ_i-corrupted and P_j (w.r.t. P_i) is τ_j-corrupted, with $\tau_i, \tau_j > \tau_0$;*
  4. *at the point of query τ_j, oracle π_j^t had a guaranteed delivery matching conversation to π_i^s, and*
  5. *$b' = \pi_i^s.b$.*

Define the advantage of an adversary $\mathcal{A}$ in the Key-Ind *security experiment $\mathsf{G}_{\Pi}^{\mathsf{Key\text{-}Ind}}(\mathcal{A})$ as*

$$\mathsf{Adv}_{\Pi}^{\mathsf{Key\text{-}Ind}}(\mathcal{A}) := \left|\Pr\left[\mathsf{G}_{\Pi}^{\mathsf{Key\text{-}Ind}}(\mathcal{A}) = 1\right] - \frac{1}{2}\right|.$$

We assume that all adversaries in the Key-Ind game are valid, meaning that they terminate and provide an output in the correct format (i.e. $(i, s, b') \in [n] \times [q] \times \{0, 1\}$). Later on in our proofs we will follow the game-hopping strategy, and in doing so we will often simplify exposition by additionally assuming adversaries that do not trigger a trivial win (in the Key-Ind game or any subsequent modifications of this game).

We define AKE security in three flavors, distinguished by the level of entity authentication that is achieved by the protocol. An adversary breaks the AKE security of a protocol if it wins either the entity authentication game, or the key indistinguishability game.

Definition 9 (Authenticated Key Exchange). *Let Π be a protocol. The advantage of an adversary $\mathcal{A}$ in terms of* AKE-M *(mutual entity authentication), resp.* AKE-I *(initiator authenticates the responder), resp.* AKE-R *(responder authenticates the initiator) is defined as follows:*

$$\mathsf{Adv}^{\mathsf{AKE\text{-}M}}_{\Pi}(\mathcal{A}) := \mathsf{Adv}^{\mathsf{Key\text{-}Ind}}_{\Pi}(\mathcal{A}) + \mathsf{Adv}^{\mathsf{Ent\text{-}Auth\text{-}I}}_{\Pi}(\mathcal{A}) + \mathsf{Adv}^{\mathsf{Ent\text{-}Auth\text{-}R}}_{\Pi}(\mathcal{A}).$$
$$\mathsf{Adv}^{\mathsf{AKE\text{-}I}}_{\Pi}(\mathcal{A}) := \mathsf{Adv}^{\mathsf{Key\text{-}Ind}}_{\Pi}(\mathcal{A}) + \mathsf{Adv}^{\mathsf{Ent\text{-}Auth\text{-}I}}_{\Pi}(\mathcal{A}).$$
$$\mathsf{Adv}^{\mathsf{AKE\text{-}R}}_{\Pi}(\mathcal{A}) := \mathsf{Adv}^{\mathsf{Key\text{-}Ind}}_{\Pi}(\mathcal{A}) + \mathsf{Adv}^{\mathsf{Ent\text{-}Auth\text{-}R}}_{\Pi}(\mathcal{A}).$$

We do not specify any protocols that provide AKE-I alone in this paper, however it is defined here for completeness.

2.3 Concurrent Execution Synchronization Robustness

We now describe a novel property for key exchange protocols. The goal is to capture, in a formal manner, how *robust* a protocol is in the event of adversarial control of the network and/or some of the parties. We seek a definition that asks: after an adversary has had control of the communication network (by executing arbitrary Send and NewSessionI/NewSessionR queries), can an honest protocol run be executed successfully? Specifically, if it is possible for the parties to lose synchronization (due to dropped messages or adversarial control) such that the parties cannot, in one protocol run, regain synchronization and compute the same key, then the protocol does not meet this property.

Our formalization follows the execution environment of the Ent-Auth and Key-Ind games described above, and allows an adversary to specify the protocol run (that it is attempting to 'interrupt') at the end of its execution by specifying two oracles. The challenger awards success if the two parties (specifically those two oracles) did not accept with the same session key. We define two flavours: a weaker version wSR in which the 'target' protocol run must be executed without any other messages interleaved, and a stronger version SR which allows arbitrary queries in between messages of the 'target' run, even to parties of the oracles involved in the 'target' run (though of course not to the two oracles).

We define an *honest protocol run* (via adversarial queries) between two oracles (with initial state `uninitialized`) as follows: a NewSessionI query was made that produced a protocol message m_1, a NewSessionR query was made to the other

oracle with input message m_1, and if this query produced a protocol message m_2 then this value was given as a Send query to the other oracle, and so on, until all protocol messages have been created and delivered, if possible. In the event that any of these queries fails (returns $\perp$) the honest protocol run aborts. This honest protocol run can be thought of as a genuine attempt to execute a protocol execution.

Definition 10 ((weak) Synchronization Robustness). *Let Π be a protocol. Let $\mathsf{G}^{\mathsf{wSR}}_{\Pi}(\mathcal{A})$ with boxed text or $\mathsf{G}^{\mathsf{SR}}_{\Pi}(\mathcal{A})$ with dashed boxed text be the following game:*

- *The challenger initializes n parties and their keys;*
- *$\mathcal{A}$ may issue queries* NewSessionI, NewSessionR *and* Send *as defined above;*
- *Once $\mathcal{A}$ has output (i, j, s, t) to indicate its conclusion, the experiment outputs 1 if and only if the following conditions hold:*
  1. *π_i^s.pid $= P_j$ and π_j^t.pid $= P_i$;*
  2. *$\pi_i^s.sk \neq \pi_j^t.sk$ or both values are $\perp$;*
  3. *an honest protocol run was executed between π_i^s and π_j^t;*
  4. *no queries were made by $\mathcal{A}$ to interrupt the protocol execution between π_i^s and π_j^t.*

  4. *no protocol messages in the transcripts of π_i^s and π_j^t were sent to any* other *oracles before they were delivered in the honest run.*

Define the advantage of an adversary $\mathcal{A}$ in the XX *security experiment $\mathsf{G}^{\mathsf{XX}}_{\Pi}(\mathcal{A})$, for $\mathsf{XX} \in \{\mathsf{wSR}, \mathsf{SR}\}$, as*

$$\mathsf{Adv}^{\mathsf{XX}}_{\Pi}(\mathcal{A}) := \Pr\left[\mathsf{G}^{\mathsf{XX}}_{\Pi}(\mathcal{A}) = 1\right].$$

Notes on the definitions. **Note 1:** Condition (4.) in the SR experiment states that for each genuine protocol message in the 'target' session, $\mathcal{A}$ must not have provided this message to any other oracles before that message is delivered as part of the 'target' run. This prevents a trivial attack where $\mathcal{A}$ delivers the final protocol message to two oracles: first to some other oracle than the 'target' oracle (but of the same party), then to the target oracle. When the (genuine) protocol message is delivered to the party for the second time the target oracle would abort. The parties have still created exactly one key for this genuine protocol run, and so condition (4.) essentially fixes the allowable output oracles as the ones that are processing protocol messages for the first time. (Replay attacks are not an issue in the wSR setting, since the execution must be uninterrupted and so any action made *after* that run has occurred has no impact on $\mathcal{A}$'s chances of winning.) **Note 2:** We do not allow Corrupt queries in this definition: in all of the protocols in this paper we assume pairwise shared key material (and specifically, no keys that are used by a party for communication with multiple other parties). This means that the adversary is not allowed to corrupt the parties in the target run with respect to each other, and that all other Corrupt queries will be of no benefit to an attacker. A similar argument follows for RevealKey queries. This

simplifies the security experiment, while capturing the property that we wish to assess. **Note 3:** In an alternative formulation of our definitions, the target protocol run could be performed by the challenger as an Execute query as seen in past literature [7]. We avoid this approach for two reasons. First, in the SR case, in order to support interleaving, the adversary would have to call the challenger to initiate each stage of the execution (i.e. $k+1$ times for a k-message protocol), and this is notationally awkward. Secondly and perhaps more importantly, our model allows the adversary to attempt to win its game in multiple protocol runs, and output the oracles which provides the best chance of success. Thus to retain the strength of the definition we would require *multiple* Execute queries, resulting in a model that looks very similar to what we have presented here.

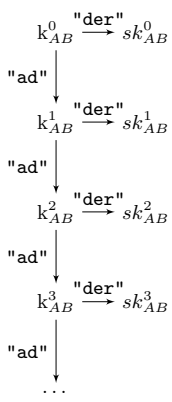
3 Linear Key Evolution

In this section we present a number of protocols that use linear key evolution to derive session keys. All of these protocols achieve wSR. It is not hard to see that full robustness (SR) is not achievable with linearly evolving protocols. To win the SR game the adversary makes a new complete protocol run after the target run has started and the session key is computed at one party, but before the session key is computed at the second party. This means that when the target session completes, the long-term key has already evolved and the key will be computed with the wrong version of the long-term key at the second party. Either the session will fail at the second party or the key will be different at the two parties. (There is a third case when the key is independent of the long-term key, but in that case the protocol fails to achieve key indistinguishability.)

The first linear key evolving protocol that we present, LP3, exchanges three short messages and has the attractive property of bounding the gap between the counters of the two parties. We present two further protocols which are even more efficient at the cost of some restrictions. LP2 is a two-message protocol but in order to maintain mutual authentication we insist that parties running LP2 have fixed their role as either initiator or responder (not an unreasonable assumption in many application scenarios). Our simplest protocol, LP1, has only a single message but, in addition to requiring fixed roles, like any other one-message protocol it can only achieve unilateral authentication. For all of our protocols we provide theorems guaranteeing authentication, key indistinguishability and weak synchronization robustness (wSR) security.

Syntax and Conventions. All protocols in this paper use message authentication codes to ensure that parties can only process messages that are meant for them. This means that party A stores a key $\mathsf{K}_{AB}^{\mathsf{MAC}}$ (static) for MAC and key derivation key $\mathsf{k}_{AB}^{\mathsf{CTR}}$ (evolving) to communicate with B, and $\mathsf{K}_{AC}^{\mathsf{MAC}}$ and $\mathsf{k}_{AC}^{\mathsf{CTR}}$ to communicate with C, etc. We describe the key derivation process in more detail in Sec. 3.1.

In LP2 and LP3, the party sending the protocol message includes its own identity in the MAC computation: this stops redirection/reflection attacks of



(a) Linear key evolution scheme.

$\mathsf{G}^{\mathsf{KEvol}}_{\mathsf{PRF}}(\mathcal{A})$

$1^\ell \xleftarrow{\$} \mathcal{A}(1^\lambda); \mathrm{k}^0 \xleftarrow{\$} \mathcal{K}_{\mathsf{PRF}}; b \xleftarrow{\$} \{0,1\}$
for $j \in \{0,\ldots,\ell\}$ **do**
  $\mathrm{k}^{j+1} = \mathsf{PRF}(\mathrm{k}^j, \texttt{"ad"})$
  $sk^j = \mathsf{PRF}(\mathrm{k}^j, \texttt{"der"})$
if $b = 1$
  $sk^* := sk^\ell$
else
  $sk^* \xleftarrow{\$} \mathcal{R}_{\mathsf{PRF}}$
$b' \xleftarrow{\$} \mathcal{A}(sk^0, \ldots, sk^{\ell-1}, sk^*, \mathrm{k}^{\ell+1})$

(b) The $\mathsf{G}^{\mathsf{KEvol}}_{\mathsf{PRF}}(\mathcal{A})$ security experiment.

Fig. 3: Linear key evolution and the corresponding experiment.

protocol messages to the sending party. For LP1 this is not necessary since the sending party advances after sending its protocol message, meaning that its state is ahead and therefore it is unable to process messages that it has already sent.

3.1 Key Derivation via Linear Evolution

Before looking at specific protocols, we define what we mean by linear key evolution and present an abstract security definition for it. Party A holds a *key derivation key* $\mathrm{k}_{AB}^{\mathsf{CTR}}$ for use in communication with party B, where the value CTR is an integer that defines the current key state, which is the number of times the key has evolved since its creation. After a party has participated in a key exchange run and computed a session key, it will apply a function Advnc to this key derivation key in order to obtain the next key derivation key and update the counter. This process is detailed in Fig. 3a. Looking ahead, forward security will be obtained if the function that computes $\mathrm{k}_{AB}^{\mathsf{CTR}+1}$ from $\mathrm{k}_{AB}^{\mathsf{CTR}}$ is one-way: this stipulation ensures that an adversary corrupting a party has no way to move upwards in the figure.

The initial "key derivation key" (KDK) is k_{AB}^0. Subsequent KDKs are derived using a pseudorandom function PRF with $\mathcal{K}_{\mathsf{PRF}} = \mathcal{R}_{\mathsf{PRF}}$ as

$$\mathrm{k}_{AB}^{i+1} = \mathsf{PRF}(\mathrm{k}_{AB}^i, \texttt{"ad"}) \tag{1}$$

and session keys are derived as

$$sk_{AB}^i = \mathsf{PRF}(\mathrm{k}_{AB}^i, \texttt{"der"})$$

where `"ad"`("advance") and `"der"`("derive") are constant labels used for domain separation.

Furthermore, for convenience, we define a function Advnc which performs multiple key derivations, if necessary. That is, $\mathsf{Advnc}(\mathrm{k}_{AB}^i, i, z)$ takes an i-th key

derivation key for some counter i and an integer z, and applies PRF iteratively z times to obtain the $(i+z)$-th KDK such that (1) is satisfied, and sets $i := i+z$. For example:

$$\mathrm{k}_{AB}^{i+z}, i+z \leftarrow \mathsf{Advnc}(\mathrm{k}_{AB}^{i}, i, z).$$

Security. For the security proofs of our protocols it will be convenient to have an abstract security definition for such a key derivation scheme, which we will show to be implied by the security of the PRF. To this end, Fig. 3b represents a security experiment for the linear key evolution scheme that we describe. The adversary $\mathcal{A}$ outputs an integer 1^ℓ (in unary, to ensure that the number ℓ is polynomially bounded for any efficient $\mathcal{A}$), and the adversary's task is to distinguish sk^ℓ from random, when given all prior session keys $sk^0, \ldots sk^{\ell-1}$ and the 'next' key derivation key $\mathrm{k}^{\ell+1}$.

Definition 11. *The advantage of $\mathcal{A}$ in in the* KEvol *security experiment defined in Fig. 3b for pseudorandom function* PRF *is defined as*

$$\mathsf{Adv}_{\mathsf{PRF}}^{\mathsf{KEvol}}(\mathcal{A}) := \left| \Pr\left[b = b'\right] - \frac{1}{2} \right|.$$

In the full version [11] we give the straightforward proof of the following theorem.

Theorem 12. *Let* PRF *be a pseudorandom function. For any adversary $\mathcal{A}$ against the* KEvol *security of* PRF*, there exists an adversary $\mathcal{B}$ against the* PRF-sec *of* PRF *such that*

$$\mathsf{Adv}_{\mathsf{PRF}}^{\mathsf{KEvol}}(\mathcal{A}) \leq \ell \cdot \mathsf{Adv}_{\mathsf{PRF}}^{\mathsf{PRF\text{-}sec}}(\mathcal{B}).$$

3.2 LP3: a Three-Message Protocol

Intuition. In Fig. 4 we present a three-message protocol called LP3, which puts a bound on how far initiator and responder can be out of sync, allows either party to initiate communications, and provides mutual authentication. After the first message is sent by an initiator, the responding party advances to catch up if they are behind. Then they respond, and the initiator does the same if they are behind. A third message confirms that both parties are now in sync again, and only after that a session key is established. We make use of state analysis proofs to show that the gap between the two states will be bounded even if messages are lost on the way (Lemma 13) and extend this proof to a scenario where concurrent runs are allowed (Lemma 14). We then show that the number of concurrent runs is a bound on the gap that can occur. We show in Theorem 15 that this also implies that the protocol achieves weak synchronization robustness (wSR). The protocol uses MACs and nonces to achieve mutual authentication (AKE-M). The functions Advnc and KDF, for PSK advancement and session key derivation respectively, are implemented using a PRF as described in Fig. 3a and Sec. 3.1.

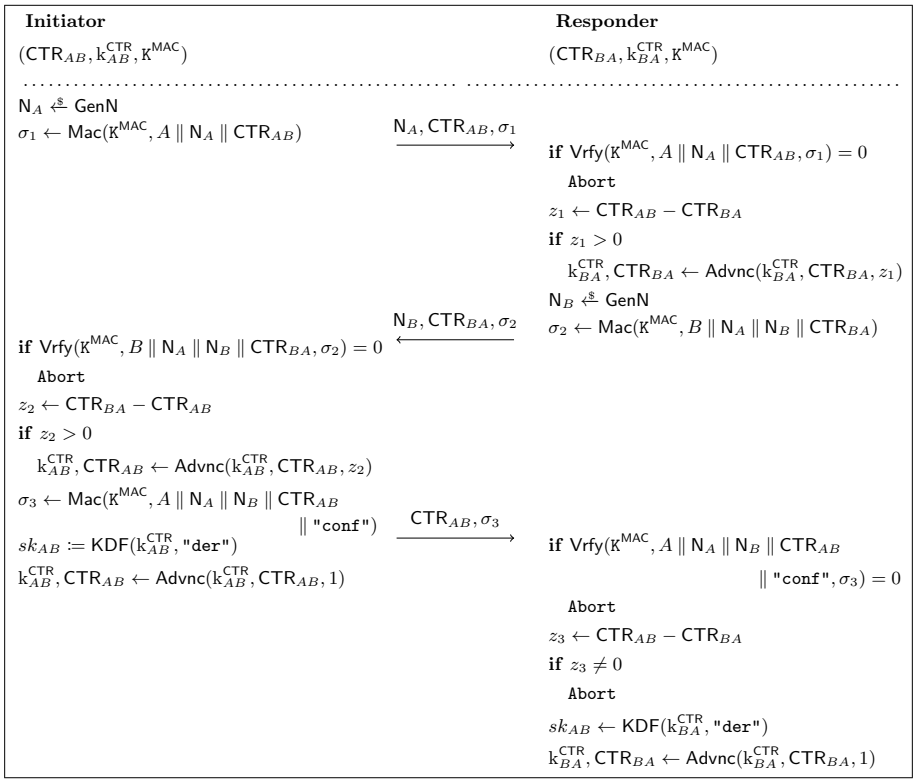


Fig. 4: LP3, a three-message protocol.

State. The protocol uses nonces on both the initiating (N_A) and responding (N_B) sides. Local session state keeps track of these, and so it is only necessary to send N_A in the first protocol message and only N_B in the second message. The nonce generation procedure is denoted GenN, and this process could be, for example, random selection of a bitstring of some fixed length, or a (per-recipient) counter maintained by the party (note however that this counter is distinct from CTR, which tracks the key derivation key's evolution stage). This choice depends on the application scenario, and this abstraction is for cleaner proofs. In the absence of a hardware RNG, random nonces require memory to be allocated for code of a software CSPRNG, while maintaining a counter requires writing to persistent storage (though such writes must be made anyway in linear key evolving protocols). The probability of a collision in random selection from $\mathcal{NS}$ can be bounded by $\mathsf{coll}[q_{\mathsf{N}}, \mathsf{GenN}] \leq \frac{q_{\mathsf{N}}^2}{2|\mathcal{NS}|}$, and the collision probability of a (per-recipient) counter of size $|\mathcal{NS}|$ that is called q_{N} times is

$$\mathsf{coll}[q_{\mathsf{N}}, \mathsf{GenN}] = \begin{cases} 0 \text{ for } 0 \leq q_{\mathsf{N}} \leq |\mathcal{NS}| - 1, \\ 1 \text{ for } q_{\mathsf{N}} \geq |\mathcal{NS}|. \end{cases}$$

We do not specify the additional counters required to make LP3 deterministic, so it is specified here as a protocol with random nonces.

LP3 achieves AKE-M security (proof in the full version [11]). The security bound is
$\mathsf{Adv}^{\mathsf{AKE\text{-}M}}_{\Pi}(\mathcal{A}) \leq n^2 \cdot \Big(4\mathsf{Adv}^{\mathsf{SEUF\text{-}CMA}\text{-}Q}_{\mathsf{MAC}}(\mathcal{B}) + 4\mathsf{coll}[q, \mathsf{GenN}] + q \cdot \mathsf{Adv}^{\mathsf{KEvol}}_{\mathsf{PRF}}(\mathcal{C})\Big)$.

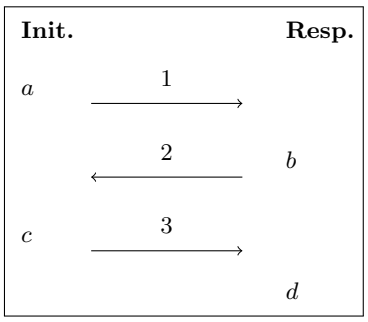
Bounded Gap: Non-Concurrent Setting. We will now prove that the "gap" between the state of the two parties in LP3 is bounded in the non-concurrent setting, that is:

Lemma 13. *Let A and B be respectively the initiator and the responder of a single — non-concurrent — LP3-run. Let δ_{AB} be the gap between A and B with respect to the evolution of the master keys of both parties. Then $\delta_{AB} \in \{-1, 0, 1\}$, assuming MAC-security.*

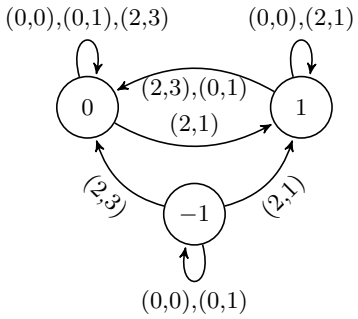
The messages in LP3 are counted in a natural way, as indicated in Fig. 5a. For this non-concurrent setting the proof is similar to [4, Lemma 1] . Then the notation "(CTR_{AB}, CTR_{BA})" means that, when the run ends, the last valid message received by A has index CTR_{AB}, and the last valid message received by B has index CTR_{BA}. We call a $(\mathsf{CTR}_{AB}, \mathsf{CTR}_{BA})$-run a run where the last message received by A is message CTR_{AB}, and the last message received by B is message CTR_{BA}. By convention $\mathsf{CTR}_{AB} = 0$ means that no message has been received by A. In Fig. 5b, we define the states to be the different values of δ_{AB}. The transitions are the possible messages. An example: if our protocol instance is in state $\delta_{AB} = -1$, and B responds to message 1 with message 2, i.e. transition $(2, 1)$ in the state diagram, the initiator will advance twice and the state will be $\delta_{AB} = 1$. A then sends the third message: transition $(2, 3)$ takes place and we end up in state $\delta_{AB} = 0$ since this third message will cause the responder to advance.

Proof. We prove Lemma 13. The protocol is initialized with $\delta_{AB} = 0$ and the first step is sending message 1: either the message never reaches the responder, or the message is received correctly. In either case neither party advances, so $\delta_{AB} = 0$ — i.e. transition $(0, 1)$ in Fig. 5b is fired. If the protocol now terminates we end up in state 0, while sending and receiving message 2 would cause the initiator to advance, or in terms of the state diagram, fire $(2, 1)$ and transition to $\delta_{AB} = 1$.

Because we restrict ourselves to non-concurrent executions, the only possible option no matter the state is to advance with one message or terminate and start from $(0, 1)$. Adding all possible transitions to the state diagram, we observe that there are no reachable states other than 0 and 1. Since the protocol does not have fixed roles we can reach a state -1 by changing roles after we reached state 1. From there, there are two transitions that bring us back to states 0 and 1. Since we assume that MACs cannot be forged, these are the only reachable states, thus $\delta_{AB} \in \{-1, 0, 1\}$ always holds.



(a) Numbering of states for the proofs of Lemmas 13 $(1, 2, 3)$ and 14 (a, b, c, d).



(b) Synchronization state for LP3 in the non-concurrent setting.

Fig. 5: Different states for LP3, and transitions between them.

Bounded Gap: Concurrent Setting. We will now extend Lemma 13 to the concurrent setting.

Lemma 14. *Let A and B be respectively the initiator and the responder of C concurrent LP3-runs. Let δ_{AB} be the gap between A and B with respect to the evolution of the master keys of both parties. Then $-C \leq \delta_{AB} \leq 1+C$, assuming MAC-security.*

To illustrate the (in a sense) multidimensional effect of concurrent runs on the protocol, we will now use a different message labelling convention. Fig. 5a defines the different states the protocol execution can be in. The state diagram in Fig. 6 now uses these four possible protocol states as diagram states — a message between state a and b is thus necessarily message 1. The internal state of the four 'macro states' in the diagram now represents the value of δ_{AB}.

Observe that for the transitions from a to b and from b to c, i.e. the sending of messages 1 and 2, respectively, the evolution of δ_{AB} depends on the actual value of a. For all transitions caused by message 3, the change is systematic:

1. Any transition from c to d will decrease δ_{AB} by 1;
2. any transition from b to c will increase δ_{AB} by at least 1.

Additionally there are two 'resets', since

3. any transition from a to b will set δ_{AB} to 0, if the gap is 1 or more;
4. any transition from b to c will set δ_{AB} to 1, if the gap is 0 or less.

Proof. We prove Lemma 14. In Lemma 13, the normal range is shown to be $\delta_{AB} \in \{-1, 0, 1\}$. Extensions beyond this range are possible when the condition in 1. or 2. above occurs during a run, so each consecutive run can influence δ_{AB} with -1 or $+1$ at most. Since we assume MAC-security, the adversary cannot influence the protocol with messages other than those authentically sent during one of the runs. Inductively, we conclude $-C \leq \delta_{AB} \leq 1 + C$.

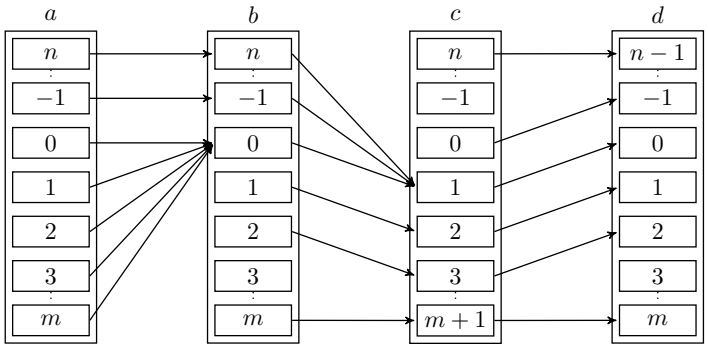


Fig. 6: Synchronization state for LP3 in the concurrent setting.

wSR of LP3. We now argue that LP3 obtains weak synchronization robustness (wSR), the property that captures how well a protocol can recover from network errors and interleaving of protocol runs. In the wSR game the adversary can make arbitrary NewSessionI, NewSessionR and Send queries, and at its conclusion it outputs the identifiers of two oracles: it is said to win the wSR game if these oracles engaged in an uninterrupted protocol run but did not compute the same session key. As such, a proof of wSR must argue that whatever values of party state exist before the target protocol run occurs, neither of the parties will abort and both will arrive at the same session key.

Our general approach for proving robustness of all of the protocols in this paper is to separate adversaries that win the wSR game via forging a MAC value, and those that do not produce a forgery during their execution. LP1 (Fig. 8) and LP2 (Fig. 7) have fixed roles and as a result the initiator's counter value must always be at least the size of the responder's counter value for the protocol to have correctness. Thus a MAC forgery can force the responding party's counter value to be arbitrarily large, and the target protocol run will cause at least one party to abort, and the adversary wins the wSR game. LP3, on the other hand, is actually not vulnerable in the sense of synchronization robustness if a MAC forgery does occur. This is due to LP3 being designed to have correctness for all starting (integer) counter values, since in any session, both parties can catch up from being arbitrarily far behind.

We formally prove this below, however to see this visually, consider Fig. 6 for the execution of a single protocol run, i.e. from a to d. For any initial state difference a, the state c after the second protocol message has been processed is always 1 (the initiator computes a session key and advances once), leading to state difference 0 after the responder processes the final protocol message (deriving a session key and advancing once).

Theorem 15 (wSR of LP3). *Let Π be the three-message protocol in Fig. 4, built using* $\mathsf{MAC} = \{\mathsf{KGen}, \mathsf{Mac}, \mathsf{Vrfy}\}$ *and* PRF *with n parties. Then for any adversary $\mathcal{A}$ against the* wSR *security of Π,* $\mathsf{Adv}^{\mathsf{wSR}}_{\Pi}(\mathcal{A}) = 0$.

Proof. The only places where `Abort` occurs in the protocol description (Fig. 4) are after MAC verification failures: in the target protocol session all messages are honestly generated so this cannot occur (assuming perfect correctness of MAC). As a result, the only route to victory in the wSR game for an adversary is to make the parties compute different session keys. This occurs if the parties compute session keys but have different counter values once all three protocol messages have been delivered and processed: following the notation and arguments in Lemma 14, this is the same as showing that $\delta = 0$ after a $(2, 3)$ session for any starting delta value. More precisely, let A and B be the parties involved in the target session where A sends the first protocol message, let $\delta_{AB}^{\mathsf{pre}}$ be the gap between A and B with respect to the evolution of the master keys of both parties and the point *before* the target session begins (i.e. before the adversary calls NewSessionI for the target session), and let $\delta_{AB}^{\mathsf{post}}$ be the gap *after* the target session has occurred. Fig. 5b shows that $\delta_{AB}^{\mathsf{post}} = 0$ for $\delta_{AB}^{\mathsf{pre}} \in \{-1, 0, -1\}$, so to complete the proof we need to show that this also holds for arbitrary $\delta_{AB}^{\mathsf{pre}}$.

If $\delta_{AB}^{\mathsf{pre}} \in \{1, 2, \ldots, \}$, i.e. CTR_{AB} is ahead of CTR_{BA} by $\delta_{AB}^{\mathsf{pre}} = z_1$ steps, then the first protocol message processing by B results in B advancing its counter CTR_{BA} by $\delta_{AB}^{\mathsf{pre}}$ steps, leading to state difference 0. This means that A will not advance on receiving the second protocol message and both parties will compute a session key for state CTR_{AB} and then advance once, and so $\delta_{AB}^{\mathsf{post}} = 0$.

If $\delta_{AB}^{\mathsf{pre}} \in \{-1, -2, \ldots, \}$, i.e. CTR_{BA} is ahead of CTR_{AB} by $-\delta_{AB}^{\mathsf{pre}} = z_2$ steps, B does not advance in processing the first message, however A does advance by $-\delta_{AB}^{\mathsf{pre}} = z_2$ steps on receiving the second protocol message. Again this leads to state difference 0 and here a session key is computed for state CTR_{BA} and then both parties advance once, so $\delta_{AB}^{\mathsf{post}} = 0$.

This concludes the proof, since any initial state will lead to the target protocol run computing the same session key for the involved parties.

3.3 LP2: A Two-Message Protocol with Fixed Roles

In Fig. 7 we present a two-message protocol, LP2, with linear key evolution. The roles of initiator and responder are fixed, so the same party initiates every session: this is enforced by $\mathsf{CTR}_{AB} \geq \mathsf{CTR}_{BA}$ (for A initiating).

Achieving weak synchronization robustness (wSR) is slightly more complicated in LP2 than it was in LP3. If we were to adapt LP3 to a two-message protocol by simply dropping the last message and having the responder accept (thus, deriving a session key and advancing its state), we could end up in a situation where we break the requirement that the responder should never advance past the state of the initiator. In this hypothetical protocol, the initiator will initiate the key exchange, but will not derive a session key until it has authenticated the responder. The responder, however, will authenticate the initiator upon receiving the first protocol message (rather than waiting for a key confirmation message as in LP3) and produce the second protocol message, after which it will immediately derive a session key and advance its state. Thus, if this second protocol message is not delivered, the responder will have advanced its state, but the initiator has not, contradicting our requirement that $\mathsf{CTR}_{AB} \geq \mathsf{CTR}_{BA}$.

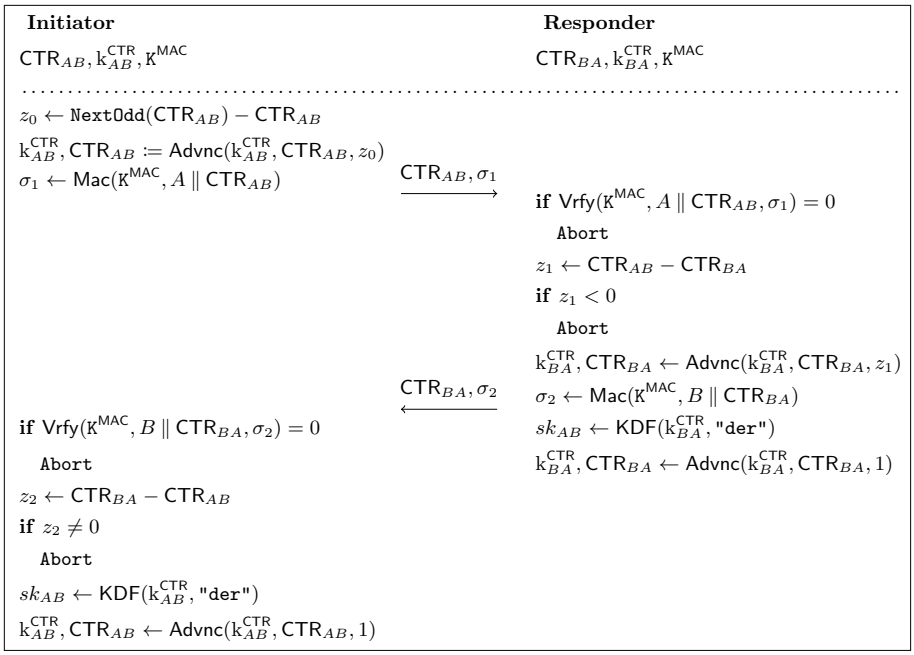


Fig. 7: LP2, a two-message protocol with fixed roles.

In order to avoid this in LP2, the initiator A will always advance to the next *odd* value of its counter at the beginning of each session. How many steps the initiator advances depends on what has happened earlier. If a complete session has been executed as A's previous action, A starts by advancing once, so that its state counter is ahead of B. If in the previous session A never processed the second protocol message, A will advance twice at the beginning of the next session, in order to catch up to B and move ahead. The reasoning behind this is the separation of A's counter set: if the counter is an even integer then A has most recently received a message (and derived a key), whereas if it is an odd integer then A most recently sent a (session opening) protocol message. In both cases, advancing to $\mathtt{NextOdd}(\mathsf{CTR}_{AB})$ will have the desired effect.

With this simpler protocol we are able to achieve most of the desired properties from SP3, but with a more lightweight protocol. Fixing the roles makes this possible, and this demonstrates the fine balance between forward security and (weak) synchronization robustness. In the event that the reduced communication complexity of LP2 compared to LP3 is desirable when choosing a protocol, but if the application demands that either party can initiate, it is possible to run LP2 in *duplex mode*. In duplex mode, both parties keep separate key derivation keys and counters for initiating and responding such that both parties can have both roles without violating the condition $\mathsf{CTR}_{AB} \geq \mathsf{CTR}_{BA}$.

LP2 provides AKE-M security, with security bound $\mathsf{Adv}^{\mathsf{AKE\text{-}M}}_{\Pi}(\mathcal{A}) \leq n^2 \cdot \Big(4 \cdot \mathsf{Adv}^{\mathsf{SEUF\text{-}CMA\text{-}}Q}_{\mathsf{MAC}}(\mathcal{B}) + q \cdot \mathsf{Adv}^{\mathsf{KEvol}}_{\mathsf{PRF}}(\mathcal{C})\Big)$. The proof [11] proceeds similarly to the LP3 proofs, except here there are no nonces so no $\mathsf{coll}[q, \mathsf{GenN}]$ term is required. LP2 also provides wSR security. The proof of this (in the full version [11]) slightly differs from LP3 because now we must additionally argue that the only way the counters can be modified is via a MAC forgery.

3.4 LP1: A One-Message Protocol with Fixed Roles

In Fig. 8 we present a one-message protocol, LP1, with linear key evolution. Like in LP2, the roles of initiator and responder are fixed, so the same party initiates every session: i.e. $\mathsf{CTR}_{AB} \geq \mathsf{CTR}_{BA}$ (for A initiating). LP1 achieves one-sided authentication (responder authenticates initiator). Achieving weak synchronization robustness (wSR) is similar in LP1 and LP2, and is guaranteed by MAC security. Theorems and proofs are in the full version [11]. Like with LP2, if both parties need to be able to initiate then LP1 can be run in *duplex mode*.

Initiator		**Responder**
$(\mathsf{CTR}_{AB}, \mathrm{k}^{\mathsf{CTR}}_{AB}, \mathrm{K}^{\mathsf{MAC}})$		$(\mathsf{CTR}_{BA}, \mathrm{k}^{\mathsf{CTR}}_{BA}, \mathrm{K}^{\mathsf{MAC}})$
$\sigma_1 \leftarrow \mathsf{Mac}(\mathrm{K}^{\mathsf{MAC}}, \mathsf{CTR}_{AB})$	$\xrightarrow{\mathsf{CTR}_{AB}, \sigma_1}$	
$sk_{AB} \leftarrow \mathsf{KDF}(\mathrm{k}^{\mathsf{CTR}}_{AB}, \texttt{"der"})$		**if** $\mathsf{Vrfy}(\mathrm{K}^{\mathsf{MAC}}, \mathsf{CTR}_{AB}, \sigma_1) = 0$
$\mathrm{k}^{\mathsf{CTR}}_{AB}, \mathsf{CTR}_{AB} \leftarrow \mathsf{Advnc}(\mathrm{k}^{\mathsf{CTR}}_{AB}, \mathsf{CTR}_{AB}, 1)$		`Abort`
		$z_1 \leftarrow \mathsf{CTR}_{AB} - \mathsf{CTR}_{BA}$
		if $z_1 < 0$
		`Abort`
		$\mathrm{k}^{\mathsf{CTR}}_{BA}, \mathsf{CTR}_{BA} \leftarrow \mathsf{Advnc}(\mathrm{k}^{\mathsf{CTR}}_{BA}, \mathsf{CTR}_{BA}, z_1)$
		$sk_{BA} \leftarrow \mathsf{KDF}(\mathrm{k}^{\mathsf{CTR}}_{BA}, \texttt{"der"})$
		$\mathrm{k}^{\mathsf{CTR}}_{BA}, \mathsf{CTR}_{BA} \leftarrow \mathsf{Advnc}(\mathrm{k}^{\mathsf{CTR}}_{BA}, \mathsf{CTR}_{BA}, 1)$

Fig. 8: LP1, a one-message protocol with fixed roles.

4 Non-Linear Key Evolution

In the previous section, we have considered protocols that deploy a linear key evolving mechanism. We have seen that the linearity of these mechanisms has significant downsides when the protocol runs multiple times in parallel between the same two parties. Especially interleaving of messages might cause all but one protocol execution to abort, which is an undesirable behavior.

In this section, we present a protocol that uses puncturable pseudorandom functions (PPRFs) as a "non-linear" key evolution mechanism. We show that

this protocol can establish many parallel sessions between two parties, while only requiring some additional storage (logarithmic in the supported maximum number sessions) and computations (in practice hash function evaluations logarithmic in the supported maximum number of sessions).

4.1 Puncturable Pseudorandom Functions

We briefly recall the basic definition of puncturable pseudorandom functions (PPRF). A PPRF is a special case of a pseudorandom function, where it is possible to compute punctured keys, which do not allow evaluation on inputs that have been punctured. We recall the definition of a PPRF and its security [29].

Definition 16 (PPRF). *A* puncturable pseudorandom function *with key space* $\mathcal{K}_{\mathsf{PPRF}}$*, domain* $\mathcal{D}_{\mathsf{PPRF}}$*, and range* $\mathcal{R}_{\mathsf{PPRF}}$ *consists of three probabilistic polynomial-time algorithms* $\mathsf{PPRF} = (\mathsf{Setup}, \mathsf{Eval}, \mathsf{Punct})$*, which are described as follows:*

- $\mathsf{Setup}(1^\lambda)$*: This algorithm takes as input the security parameter* λ *and outputs a description of a key* $k \in \mathcal{K}_{\mathsf{PPRF}}$*.*
- $\mathsf{Eval}(k, x)$*: This algorithm takes as input a key* $k \in \mathcal{K}_{\mathsf{PPRF}}$ *and a value* $x \in \mathcal{D}_{\mathsf{PPRF}}$*, and outputs a value* $y \in \mathcal{R}_{\mathsf{PPRF}}$*, or a failure symbol* $\perp$*.*
- $\mathsf{Punct}(k, x)$*: This algorithm takes as input a key* $k \in \mathcal{K}_{\mathsf{PPRF}}$ *and a value* $x \in \mathcal{D}_{\mathsf{PPRF}}$*, and returns a punctured key* $k' \in \mathcal{K}_{\mathsf{PPRF}}$*.*

Note that the puncturing procedure can also output an unmodified key (i.e. $k' = k$). This is for example reasonable if the procedure is called on an already-punctured value.

Definition 17 (PPRF Correctness). *A PPRF is* correct *if for every subset* $\{x_1, \ldots, x_t\} = \mathcal{S} \subseteq \mathcal{D}_{\mathsf{PPRF}}$ *and all* $x \in \mathcal{D}_{\mathsf{PPRF}} \setminus \mathcal{S}$*, it holds that*

$$\Pr\left[\mathsf{Eval}(k_0, x) = \mathsf{Eval}(k_t, x) : \begin{array}{c} k_0 \xleftarrow{\$} \mathsf{Setup}(1^\lambda); \\ k_i = \mathsf{Punct}(k_{i-1}, x_i) \text{ for } i \in [t]; \end{array}\right] = 1.$$

The security experiment asks that an adversary cannot distinguish an evalution of a real input (provided by the adversary) from a random output range element, even if the adversary has access to an evaluation oracle and the key that results from puncturing on the challenge input.

Definition 18 (PPRF Security). *The advantage of an adversary* $\mathcal{A}$ *in the* rand *security experiment* $\mathsf{G}^{\mathsf{rand}}_{\mathsf{PPRF}}(\mathcal{A})$ *defined in Fig. 9 is*

$$\mathsf{Adv}^{\mathsf{rand}}_{\mathsf{PPRF}}(\mathcal{A}) := \left|\Pr\left[\mathsf{G}^{\mathsf{rand}}_{\mathsf{PPRF}}(\mathcal{A}) = 1\right] - \frac{1}{2}\right|.$$

$\mathsf{G}^{\mathsf{rand}}_{\mathsf{PPRF}}(\mathcal{A})$

$k \xleftarrow{\$} \mathsf{Setup}(1^\lambda)$
$b \xleftarrow{\$} \{0,1\};\ \mathcal{Q} := \emptyset$
$x^* \xleftarrow{\$} \mathcal{A}^{\mathcal{O}_{\mathsf{Eval}}(\cdot)}(1^\lambda)$
$y_0 \xleftarrow{\$} \mathcal{R}_{\mathsf{PPRF}};\ y_1 \leftarrow \mathsf{Eval}(k, x^*)$
$k \leftarrow \mathsf{Punct}(k, x^*)$
$b^* \xleftarrow{\$} \mathcal{A}(k, y_b)$
return 1 **if** $b = b^*$ **and** $x^* \notin \mathcal{Q}$
return 0

$\mathcal{O}_{\mathsf{Eval}}(x)$

$y \leftarrow \mathsf{Eval}(k, x)$
$\mathcal{Q} := \mathcal{Q} \cup \{x\}$
$k \leftarrow \mathsf{Punct}(k, x)$
return y

Fig. 9: The rand security experiment for puncturable PRF PPRF.

4.2 PPRF-based Symmetric AKE

Intuition. The main idea of our PPRF-based protocol is to derive the session key via an evaluation of the PPRF. That is, both parties share a PPRF evaluation key k, which is used to derive session keys by computing $\mathsf{Eval}(\mathrm{k}, \mathsf{N}_A)$ for some value N_A (in our protocols this will be a counter). After derivation of a session key, the PPRF key will also be punctured at the value N_A by computing $\mathrm{k} \leftarrow \mathsf{Punct}(\mathrm{k}, \mathsf{N}_A)$. Note that the new key k cannot recompute $\mathsf{Eval}(\mathrm{k}, \mathsf{N}_A)$ as it has been punctured for N_A. This will be our leverage to achieve forward security.

Additionally, the PPRF is an essential building block to achieve full synchronization robustness in our protocols. Intuitively, the puncturing procedure of a PPRF does not evolve its key "linearly" but rather enables fine-grained removal of evaluation capabilities. This guarantees that every protocol run with some fresh value N_A for $\mathsf{Eval}(\mathrm{k}, \mathsf{N}_A)$ will be completed successfully, even if other protocol runs with some value $\mathsf{N}'_A \neq \mathsf{N}_A$ are executed in-between.

Our protocols. We present a one-message and a two-message protocol, based on PPRFs. Both protocols have fixed roles, meaning the same party will always initiate (and only this party is required to store the counter). The two-message protocol implicitly authenticates both parties (and thus achieves mutual authentication), while the one-message protocol inherently only achieves responder-only authentication (responder authenticates initiator). Hence, we will only focus on the two-message protocol shown in Fig. 10 and provide a description and security analysis for the one-message protocol in the full version [11].

Another important aspect of our protocols is that they use counters to systematically "exhaust" the PPRF. We will later discuss that this approach assists the efficiency of tree-based PPRFs as discussed in Aviram et al. [3]. The number of session keys that can be derived is equal to the size of the counter space.

In the full version [11] we prove that protocol PP2 shown in Fig. 10 provides key indistinguishability and mutual authentication. The reduction is standard, and we require SEUF-CMA-Q of the MAC and rand security of the puncturable PRF.

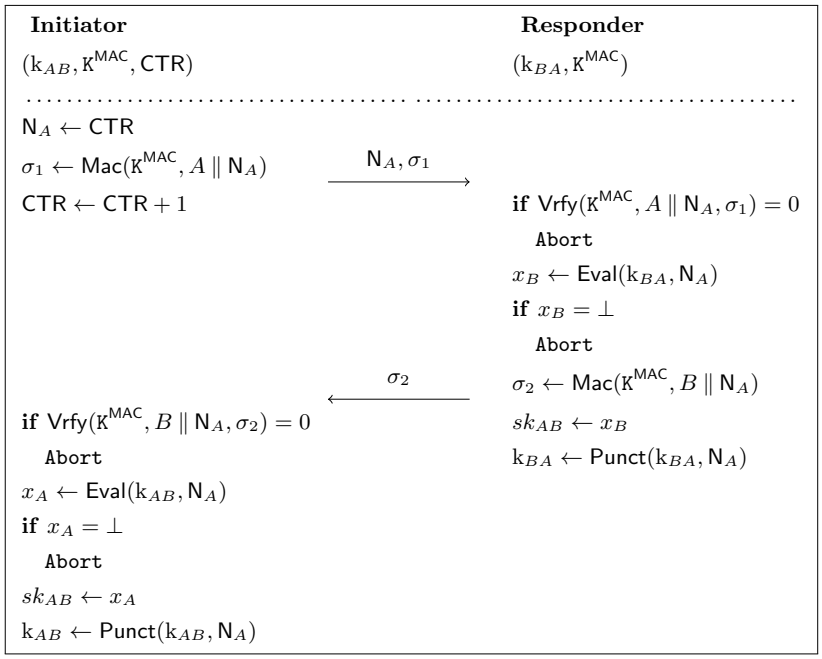


Fig. 10: A symmetric AKE protocol PP2 that tolerates concurrent sessions, using a puncturable PRF $\mathsf{PPRF} = (\mathsf{Setup}, \mathsf{Eval}, \mathsf{Punct})$.

4.3 Synchronization Robustness of PP2

In the full version [11] we prove that PP2 achieves full synchronization robustness (SR) with security bound $\mathsf{Adv}^{\mathsf{SR}}_{\Pi}(\mathcal{A}) \leq n^2 \cdot \mathsf{Adv}^{\mathsf{SEUF\text{-}CMA}\text{-}Q}_{\mathsf{MAC}}(\mathcal{B})$, and here we give a high-level overview of the proof strategy. Intuitively we want to show that any adversary, making arbitrary message delivery queries between any of the parties (and their session oracles), cannot cause an adversarially chosen but honestly executed target protocol run to break down.

The robustness proof essentially needs three arguments: 1) the adversary cannot forge protocol messages without breaking the security of the MAC, 2) replaying messages from the target protocol run to other oracles is not beneficial to the adversary, and 3) the correctness of the PPRF ensures that interleaving queries with nonce values different to the one used in the target session will not influence the successful computation of a session key in the target session.

4.4 Instantiation

It remains to discuss how PP2 can be instantiated with a PPRF and what impact the PPRF has on its efficiency. A promising candidate is the Goldreich–Goldwasser–Micali PRF [19], which can be transformed to a PPRF [10, 13, 24]. We give an intuitive explanation of the construction and refer the reader to [3] for

a more detailed description and analysis. This construction is especially suitable, as both the PPRF evaluation and puncturing are solely based on hash function evaluations in practice.

Intuition. The tree-based PPRF uses two functions H_0 and H_1 both mapping from $\{0,1\}^\lambda$ to $\{0,1\}^\lambda$. For every input $x \in \{0,1\}^\lambda$ of the PPRF, the binary representation of x prescribes the sequence in which H_0 and H_1 have to be repeatedly applied to x. For example, $\mathsf{Eval}(01) = H_1(H_0(x))$. Note that the evaluation of x corresponds to a path through a binary tree, where each bit in x tells you whether to take a "left" or "right" path. The result of an evaluation always corresponds to a leaf in the binary tree.

The initial PPRF key consists of the root node, which is initialized during key generation as a randomly chosen string. To puncture values (i.e., to puncture leaves of the tree), we precompute and store all nodes on the co-path between the root and the leaf, before deleting all parent nodes (including the root node) of the leaf. Note that this procedure can be repeated for any of the leaves and note that it satisfies all puncturing-relevant properties (i.e., re-computation of $\mathsf{Eval}(x)$ is not possible but the correctness of the PPRF remains intact).

Memory Consumption. We briefly discuss the memory consumed by the PPRF during the lifetime of PP2 (and PP1). First, note that the PPRF-based protocols deploy counters, which (if all messages are delivered in sequence) ensure a systematic puncturing from the leftmost leaf to the rightmost leaf of the binary tree. This yields the need to to store at most $\log(|\mathsf{CTR}|)$ tree nodes (i.e., at most one node per layer of the tree) at any point in time. For C concurrent sessions, this bound increases to a maximum of $C \cdot \log(|\mathsf{CTR}|)$ tree nodes.

The analysis gets slightly more difficult if an adversary *actively drops* protocols messages. Each dropped message will either cause the initiator or both parties to not puncture at some position. One approach to tame the memory consumption in this case, would be to always puncture on all values which are smaller than $\mathsf{CTR}-C$.[3] As we never expect more than C sessions in parallel, this reduces additional memory caused by lost messages. In this case, the memory consumption is again upper-bounded by $C \cdot \log(|\mathsf{CTR}|)$ tree nodes.

Finally, note that in the one-message protocol PP1 [11] the initiator always punctures strictly in order and thus has to store at most $\log(|\mathsf{CTR}|)$ tree nodes. This may be particularly useful in an application where many low-end devices communicate with a central server.

References

1. Alwen, J., Coretti, S., Dodis, Y.: The double ratchet: Security notions, proofs, and modularization for the Signal protocol. In: Ishai, Y., Rijmen, V. (eds.) EURO-

[3] Interestingly, the tree-based PPRF can puncture multiple values in one go by "chopping off" whole branches of the tree, instead of puncturing all values one after another.

CRYPT 2019, Part I. LNCS, vol. 11476, pp. 129–158. Springer, Heidelberg (May 2019). 10.1007/978-3-030-17653-2_5
2. Retail Financial Services Symmetric Key Management Part 1: Using Symmetric Techniques (ANSI x9.24). Standard, American National Standards Institute, New York, USA (2009)
3. Aviram, N., Gellert, K., Jager, T.: Session resumption protocols and efficient forward security for TLS 1.3 0-RTT. In: Ishai, Y., Rijmen, V. (eds.) EUROCRYPT 2019, Part II. LNCS, vol. 11477, pp. 117–150. Springer, Heidelberg (May 2019). 10.1007/978-3-030-17656-3_5
4. Avoine, G., Canard, S., Ferreira, L.: Symmetric-key authenticated key exchange (SAKE) with perfect forward secrecy. In: Jarecki, S. (ed.) CT-RSA 2020. LNCS, vol. 12006, pp. 199–224. Springer, Heidelberg (Feb 2020). 10.1007/978-3-030-40186-3_10
5. Bellare, M., Goldreich, O., Mityagin, A.: The power of verification queries in message authentication and authenticated encryption. Cryptology ePrint Archive, Report 2004/309 (2004), http://eprint.iacr.org/2004/309
6. Bellare, M., Miner, S.K.: A forward-secure digital signature scheme. In: Wiener, M.J. (ed.) CRYPTO'99. LNCS, vol. 1666, pp. 431–448. Springer, Heidelberg (Aug 1999). 10.1007/3-540-48405-1_28
7. Bellare, M., Pointcheval, D., Rogaway, P.: Authenticated key exchange secure against dictionary attacks. In: Preneel, B. (ed.) EUROCRYPT 2000. LNCS, vol. 1807, pp. 139–155. Springer, Heidelberg (May 2000). 10.1007/3-540-45539-6_11
8. Bellare, M., Rogaway, P.: Entity authentication and key distribution. In: Stinson, D.R. (ed.) CRYPTO'93. LNCS, vol. 773, pp. 232–249. Springer, Heidelberg (Aug 1994). 10.1007/3-540-48329-2_21
9. Bellare, M., Yee, B.S.: Forward-security in private-key cryptography. In: Joye, M. (ed.) CT-RSA 2003. LNCS, vol. 2612, pp. 1–18. Springer, Heidelberg (Apr 2003). 10.1007/3-540-36563-X_1
10. Boneh, D., Waters, B.: Constrained pseudorandom functions and their applications. In: Sako, K., Sarkar, P. (eds.) ASIACRYPT 2013, Part II. LNCS, vol. 8270, pp. 280–300. Springer, Heidelberg (Dec 2013). 10.1007/978-3-642-42045-0_15
11. Boyd, C., Davies, G.T., de Kock, B., Gellert, K., Jager, T., Millerjord, L.: Symmetric key exchange with full forward security and robust synchronization. IACR Cryptol. ePrint Arch. p. 702 (2021), https://eprint.iacr.org/2021/702
12. Boyd, C., Gellert, K.: A Modern View on Forward Security. The Computer Journal (08 2020). 10.1093/comjnl/bxaa104, https://doi.org/10.1093/comjnl/bxaa104, https://doi.org/10.1093/comjnl/bxaa104
13. Boyle, E., Goldwasser, S., Ivan, I.: Functional signatures and pseudorandom functions. In: Krawczyk, H. (ed.) PKC 2014. LNCS, vol. 8383, pp. 501–519. Springer, Heidelberg (Mar 2014). 10.1007/978-3-642-54631-0_29
14. Brier, E., Peyrin, T.: A forward-secure symmetric-key derivation protocol - how to improve classical DUKPT. In: Abe, M. (ed.) ASIACRYPT 2010. LNCS, vol. 6477, pp. 250–267. Springer, Heidelberg (Dec 2010). 10.1007/978-3-642-17373-8_15
15. Canetti, R., Krawczyk, H.: Analysis of key-exchange protocols and their use for building secure channels. In: Pfitzmann, B. (ed.) EUROCRYPT 2001. LNCS, vol. 2045, pp. 453–474. Springer, Heidelberg (May 2001). 10.1007/3-540-44987-6_28
16. Cini, V., Ramacher, S., Slamanig, D., Striecks, C.: CCA-secure (puncturable) KEMs from encryption with non-negligible decryption errors. In: Moriai, S., Wang,

H. (eds.) ASIACRYPT 2020, Part I. LNCS, vol. 12491, pp. 159–190. Springer, Heidelberg (Dec 2020). 10.1007/978-3-030-64837-4_6
17. Derler, D., Jager, T., Slamanig, D., Striecks, C.: Bloom filter encryption and applications to efficient forward-secret 0-RTT key exchange. In: Nielsen, J.B., Rijmen, V. (eds.) EUROCRYPT 2018, Part III. LNCS, vol. 10822, pp. 425–455. Springer, Heidelberg (Apr / May 2018). 10.1007/978-3-319-78372-7_14
18. Dousti, M.S., Jalili, R.: FORSAKES: A forward-secure authenticated key exchange protocol based on symmetric key-evolving schemes. Adv. Math. Commun. **9**(4), 471–514 (2015). 10.3934/amc.2015.9.471
19. Goldreich, O., Goldwasser, S., Micali, S.: How to construct random functions. Journal of the ACM **33**(4), 792–807 (Oct 1986)
20. Green, M.D., Miers, I.: Forward secure asynchronous messaging from puncturable encryption. In: 2015 IEEE Symposium on Security and Privacy. pp. 305–320. IEEE Computer Society Press (May 2015). 10.1109/SP.2015.26
21. Günther, F., Hale, B., Jager, T., Lauer, S.: 0-RTT key exchange with full forward secrecy. In: Coron, J.S., Nielsen, J.B. (eds.) EUROCRYPT 2017, Part III. LNCS, vol. 10212, pp. 519–548. Springer, Heidelberg (Apr / May 2017). 10.1007/978-3-319-56617-7_18
22. FIPS 198-1. the Keyed-Hash Message Authentication Code (HMAC). Standard, NIST (2008)
23. Jager, T., Kohlar, F., Schäge, S., Schwenk, J.: On the security of TLS-DHE in the standard model. In: Safavi-Naini, R., Canetti, R. (eds.) CRYPTO 2012. LNCS, vol. 7417, pp. 273–293. Springer, Heidelberg (Aug 2012). 10.1007/978-3-642-32009-5_17
24. Kiayias, A., Papadopoulos, S., Triandopoulos, N., Zacharias, T.: Delegatable pseudorandom functions and applications. In: Sadeghi, A.R., Gligor, V.D., Yung, M. (eds.) ACM CCS 2013. pp. 669–684. ACM Press (Nov 2013). 10.1145/2508859.2516668
25. (NIST SP)-800-185. SHA-3 derived functions: cSHAKE, KMAC, TupleHash and ParallelHash. Special Publication. Standard, NIST (2016)
26. Krawczyk, H.: HMQV: A high-performance secure Diffie-Hellman protocol. In: Shoup, V. (ed.) CRYPTO 2005. LNCS, vol. 3621, pp. 546–566. Springer, Heidelberg (Aug 2005). 10.1007/11535218_33
27. Le, T.V., Burmester, M., de Medeiros, B.: Universally composable and forward-secure RFID authentication and authenticated key exchange. In: Bao, F., Miller, S. (eds.) ASIACCS 07. pp. 242–252. ACM Press (Mar 2007)
28. Li, Y., Schäge, S., Yang, Z., Kohlar, F., Schwenk, J.: On the security of the pre-shared key ciphersuites of TLS. In: Krawczyk, H. (ed.) PKC 2014. LNCS, vol. 8383, pp. 669–684. Springer, Heidelberg (Mar 2014). 10.1007/978-3-642-54631-0_38
29. Sahai, A., Waters, B.: How to use indistinguishability obfuscation: deniable encryption, and more. In: Shmoys, D.B. (ed.) 46th ACM STOC. pp. 475–484. ACM Press (May / Jun 2014). 10.1145/2591796.2591825
30. Sun, S., Sakzad, A., Steinfeld, R., Liu, J.K., Gu, D.: Public-key puncturable encryption: Modular and compact constructions. In: Kiayias, A., Kohlweiss, M., Wallden, P., Zikas, V. (eds.) PKC 2020, Part I. LNCS, vol. 12110, pp. 309–338. Springer, Heidelberg (May 2020). 10.1007/978-3-030-45374-9_11