On Password-Authenticated Key Exchange Security Modeling[⋆]

Jean Lancrenon

Université du Luxembourg,
Interdisciplinary Centre for Security, Reliability and Trust,
6 rue Richard Coudenhove-Kalergi, L-1359 Luxembourg City,
LUXEMBOURG
`jean.lancrenon@uni.lu`

Abstract. Deciding which security model is the right one for Authenticated Key Exchange (AKE) is well-known to be a difficult problem. In this paper, we examine definitions of security for Password-AKE (PAKE) in the style proposed by Bellare et al. [5] at *Eurocrypt 2000*. Indeed, there does not seem to be any consensus, even when narrowing the study down to this particular authentication method and model style, on how to precisely define fundamental notions such as accepting, terminating, and partnering. The aim of this paper is to begin addressing this problem. We first show how definitions vary from paper to paper. We then propose and thoroughly motivate a definition of our own, and use the opportunity to correct a minor flaw in a more recent and more PAKE-appropriate model proposed by Abdalla et al. [3] at *Public Key Cryptography 2005*. Finally, we argue that the uniqueness of partners holding with overwhelming probability ought to be an explicitly required and proven property for AKE in general, but even more so in the password case, where the optimal security bound one aims to achieve is no longer a negligible value. To drive this last point, we exhibit a protocol that is provably secure following the Abdalla et al. definition, and at the same time fails to satisfy this property.

1 Introduction

Password-authenticated key exchange. *Key Exchange* (KE) is preoccupied with establishing a secure ephemeral *session key* between two remote parties over an insecure network. The well-known Diffie-Hellman protocol [16] solves this problem over communication lines that are perhaps eavesdropped on, but remain otherwise undisturbed. *Authenticated* KE (AKE) aims to realize KE in such a way that the communicating parties gain some guarantee that they have established a key with the right partner even in the face of malicious - i.e. more than just eavesdropping - attackers. This is only possible if all honest parties in play have some sort of trusted setup in place prior to the exchange, taking the form e.g. of a Public Key Infrastructure (PKI) correctly managing everybody's public keying information, or pairwise-Shared long-term Symmetric Keys (SSK), etc. *Password* AKE (PAKE) is AKE in which the long-term keys are simple passwords.

Using ordinary passwords - by which we mean low-entropy, human-memorable pieces of data - as long-term authentication material cannot simply be securely viewed as a particular case of the shared-symmetric-key scenario. Indeed, the low entropy of passwords makes them vulnerable to dictionary attacks, both off-line and on-line. Hence, special care must be taken in designing a PAKE protocol. Note that it is always trivially

[⋆] This is the full version of paper [30], which appeared in the proceedings of the *PASSWORDS 2015* workshop in Cambridge, UK, in December 2015.

possible to simply run the protocol with an honest participant using a guessed password as input, and observing whether the exchange fails or succeeds. Such an on-line attack is inherent to the service, and cannot be avoided. Intuitively, we would like to limit a network adversary's effective attacks to this trivial strategy. This implies namely that untampered protocol network traffic must protect, at least computationally, every single bit of information on the underlying passwords. In a nutshell, on-line attacks are limited to successive login attempts, and off-line attacks are completely prohibited.

Security models. Complexity theoretic PAKE security models capturing these requirements made their first appearance at Eurocrypt 2000, in the form of the Bellare et al. [5] and Boyko et al. [9] models. This paper is concerned with proofs of security using the former, which we shall call BPR-style proofs. We also consider proofs using Abdalla et al.'s [3] model, which is a slight modification of the [5] model. BPR-style proofs are by far the most common in the literature, but unfortunately they are not very easy to understand. One reason for this is that often, small technicalities vary in the models' definitions from paper to paper, and it is not always clear why. This is especially the case when trying to define *when it is that two parties have had a correct protocol conversation*, a notion known as *partnering.*

Our aim is to draw attention to this issue, in order to at least start clarifying it. This is important to be able to more accurately interpret security claims in papers.

Our contributions. In Section 2, we focus on the current state of BPR-style modeling in the literature. We first show how definitions vary from paper to paper in terms of instances accepting, terminating, and partnering, citing precise examples. We then show how the subsequent security model of Abdalla et al. [3] is slightly bugged. These considerations lead into Section 3, in which we redefine the model(s) to fit all usual protocol scenarios and also fix the bug in [3]. The change has to do with the uniqueness requirement of the partner in the definition. As a follow-up, we examine this uniqueness property in detail in Section 4. We namely present evidence that it ought to be an explicitly stated and proven property of *any* AKE protocol, and even more so in the PAKE case, where the optimal security bounds are no longer negligible values (in order to account for the adversary's natural ability to simply guess a password and try it out). To this end, we exhibit a PAKE which is provably secure according to the Abdalla et al. definition (the proof is in Appendix A), but fails to satisfy that partners be unique with overwhelming probability.

Related work. PAKE was first considered by Bellovin and Merritt [7] and Jablon [21], with informal security arguments. Lucks [31] and Halevi et al. [20] were the first to produce formal models involving passwords. They were followed by Bellare et al. [5] and Boyko et al. [9], building on the AKE models introduced in [6] and [35], respectively. Since, many protocols have been proposed and studied: Katz et al. [23] showed that PAKE can be practically realized without random oracles but with a *Common Reference String* (CRS), Goldreich et al. [18] showed that PAKE can be realized solely under general complexity assumptions, and Canetti et al. [13] introduced *universally composable* (see [12]) PAKE, to name a few. A more complete bibliography on PAKE can be found in [34].

Some research on comparing AKE security models in general has been conducted. Notably, Choo et al. [14] and Cremers [15] examine how differing definitions affect security in indistinguishability-based models, but none of these works focus on password protocols. To our knowledge, nothing of the sort for PAKE has been considered yet.

Organization of the paper. The rest of the paper is structured as follows. Section 2 overviews existing PAKE model definitions, and points to some differences and inconsistencies. Next, Section 3 revisits all of these definitions, trying to place them under one roof, and introduces the matter of partner uniqueness. Then, Section 4 shows why making uniqueness of partners a property is important, especially in the context of PAKEs. Finally, Section 5 concludes the paper.

2 Different BPR-style models

In what follows, we denote $\perp$ the special error symbol and ε the empty string. We use $\mathcal{U}$ to designate both the entity $\mathcal{U}$, and the bitstring identifying $\mathcal{U}$.

2.1 The models' main foundations

Here we describe the main aspects commonly shared by all BPR-style models. Some of the notions are *at first* left intentionally vague. We first detail the parties in play, then we list the adversary's abilities. The security definitions capturing confidentiality of SKs and authentication come at the end.

Principals and instances. An interactive game, indexed by the security parameter $\lambda \in \mathbb{N}$, is played between a challenger $\mathcal{CH}$ and an adversary $\mathcal{A}$. All of the algorithms considered are Probabilistic, Polynomial-Time (PPT) in λ.

At the beginning of the game, there is a fixed set of *principals* (or *users*), which is partitioned into non-empty sets of *clients* $\mathcal{C}$ and *servers* $\mathcal{S}$. Each client $\mathcal{C}$ is assigned a *password* $pw_{\mathcal{C}}$ drawn uniformly at random from some finite - possibly small - set PW of bitstrings. Each server $\mathcal{S}$ holds the full set of all clients' passwords $\{pw_{\mathcal{C}}\}_{\mathcal{C}}$.

The adversary $\mathcal{A}$ has oracle access - via the queries described below - to any number of *instances* $\mathcal{U}^i$ ($i \in \mathbb{N}$) of any principal $\mathcal{U}$. Intuitively, an instance of $\mathcal{U}^i$ represents an attempt $\mathcal{U}$ makes at running the AKE protocol over the network, which is fully controlled by $\mathcal{A}$. An instance's objective is to compute a *Session Key* (or SK) it believes it shares with an instance $\mathcal{V}^j$ of some other principal $\mathcal{V}$. This should happen only if $\mathcal{U}^i$ thinks it has *"had a correct exchange with"* $\mathcal{V}^j$.

At any point in time, an instance $\mathcal{U}^i$ may declare itself *"ready to use a SK"*. By this time, the instance should have computed **1)** a *Partner Identity* (or PID) $pid^i_{\mathcal{U}}$, a **2)** *Session Identity* (SID) $sid^i_{\mathcal{U}}$, and of course **3)** the SK $sk^i_{\mathcal{U}}$ itself. The PID is a bitstring indicating the identity of the instance with which $\mathcal{U}^i$ believes it has communicated with. The SID is a bitstring serving as an identifier for both the key exchange run that just occurred, and the session in which the computed SK will subsequently serve. Often in practice the SID is set to being the ordered concatenation of all exchanged protocol messages, except possibly the last message. At any point in time an instance may also *"refuse to participate any longer in the protocol"*, and halt altogether. This can happen if a message e.g has an incorrect format, or if authentication fails. Once an instance has finished - either declaring it has a SK, or having just stopped - it can no longer be reused.

We purposefully did not make the notions of "having a correct exchange with an instance", "declaring oneself ready to use a SK", and "refusing to participate any longer in the protocol" precise, because these are where consensus does not seem to hold. They will be made more precise **1)** in the different examples we give further below and **2)** when we give and motivate our own definition in Section 3. These notions are also correlated to when and how the PID, SID, and SK are computed.

We now describe the oracle queries the adversary has access to. These change according to whether we are in the *Find-then-Guess* (or FtG) model [5] or the *Real-or-Random* (or RoR) model [3].[1] The security definitions themselves will come after.

Find-then-Guess. This is the original model introduced in [5]. It can be viewed as a "password spin-off" of the classic Bellare/Rogaway model [6] for AKE security.

- $\mathsf{send}(\mathcal{U}, i, m)$: $\mathcal{A}$ has message m delivered to $\mathcal{U}^i$. $\mathcal{U}^i$ processes the message according to protocol specification. To instruct an instance $\mathcal{U}$ to send the first protocol message to entity $\mathcal{V}$, $\mathcal{A}$ makes the query with $m = \mathcal{V}$. This query is used to model arbitrary message delivery to an instance. In particular, it serves to count impersonation attacks.
- $\mathsf{execute}(\mathcal{U}, \mathcal{V}, i, j)$: The protocol is executed faithfully and completely between $\mathcal{U}^i$ and $\mathcal{V}^j$ and the resulting transcript is given to $\mathcal{A}$. $\mathcal{A}$ thus gets to see as many honest protocol runs as it wishes.
- $\mathsf{reveal}(\mathcal{U}, i)$: If $\mathcal{U}^i$ is not ready to use a SK, the query returns $\perp$. Otherwise, it returns $sk^i_{\mathcal{U}}$ to $\mathcal{A}$. This models leakage of session key information through use in the ensuing session (the quality of the algorithms of which we know *nothing* about).
- $\mathsf{test}(\mathcal{U}, i)$: If $\mathcal{U}^i$ is not ready to use a SK, this returns $\perp$. Otherwise, $\mathcal{CH}$ flips a coin b outside of $\mathcal{A}$'s view. If $b = 0$, a random string R is drawn from the session key space and $tk^i_{\mathcal{U}} \leftarrow R$. Otherwise, $tk^i_{\mathcal{U}} \leftarrow sk^i_{\mathcal{U}}$. The *test key* (or TK) $tk^i_{\mathcal{U}}$ is returned to $\mathcal{A}$. The test query may only be used once in the game. Its purpose is to measure the adversary's advantage in breaking session key security.

Eventually, $\mathcal{A}$ halts the overall game, at which point it outputs a bit b'. If the game was halted without $\mathcal{A}$ making any test query, then $\mathcal{CH}$ privately flips a coin b.

Freshness: To get a meaningful definition of SK security (see below), a restriction must be put in place on the test query. Namely, $\mathcal{A}$ cannot be considered victorious if it already trivially knows the SK that it tested. Therefore, the notion of *freshness* must be introduced: An instance $\mathcal{U}^i$ is *fresh* if neither it, nor any instance with which it has had a correct exchange, has been the subject of a reveal query. *Thus, we must slightly modify the* test *and* reveal *queries presented above:* we add that test returns $\perp$ if $\mathcal{U}^i$ is not fresh and that reveal returns $\perp$ if $\mathcal{U}^i$, or an instance with which it has had a correct exchange, has been tested.

Real-or-Random. This is the model introduced in [3]. In a nutshell, it differs from FtG in that it makes all of the keys revealed to the adversary either the truly computed keys or completely random strings. In [3] it is proven that for PAKEs, this makes a real security difference. Thus, it is recommended to use RoR rather than FtG when employing BPR-style models. (See [3] for details.)

The send and $\mathsf{execute}$ queries are identical to those in the FtG model. However, the reveal query is no longer available. Instead, it is replaced by as many test queries as $\mathcal{A}$ wants. How they are answered depends on the value of a bit b flipped by $\mathcal{CH}$ outside of $\mathcal{A}$'s view *at the beginning of the game.*

- $\mathsf{test}(\mathcal{U}, i)$: If $\mathcal{U}^i$ is not ready to use a session key, $\perp$ is returned. Otherwise, suppose first that $b = 0$. If $\mathcal{U}^i$ has not had a correct exchange with any instance, or no instance that it has had a correct exchange with was subjected to a test query, $\mathcal{CH}$ selects a random R from the session key space and sets $tk^i_{\mathcal{U}} \leftarrow R$. If $\mathcal{U}^i$ has had a correct exchange with an instance $\mathcal{V}^j$ that has been tested, $\mathcal{CH}$ sets $tk^i_{\mathcal{U}} \leftarrow tk^j_{\mathcal{V}}$. Suppose now that $b = 1$. In this case, $\mathcal{CH}$ sets $tk^i_{\mathcal{U}} \leftarrow sk^i_{\mathcal{U}}$. Then, $\mathcal{A}$ receives $tk^i_{\mathcal{U}}$.

[1] To simplify our exposition, in this preliminary study we make no attempt at dealing with the *corruption* query - used to model the important property of *forward secrecy* - in this paper.

The slight complication arising in case $b = 0$ is that even if the SKs assigned are random, they must at least remain consistent across instances that should hold the same keys. As in FtG, at any point in time $\mathcal{A}$ may halt the game and output a bit b'.

Technically defining security. In both FtG and RoR, one usually considers three security properties: SK security, Client-to-Server (C2S) authentication, and Server-to-Client (S2C) authentication.

SK security: Let S be the event that "$b' = b$ at the end of the game". Event S measures the *semantic security of the SK*, i.e. the adversary's ability to tell this key apart from a random string. Since $\mathcal{A}$ can simply flip a coin to get the answer right with probability 1/2, $\mathcal{A}$'s natural advantage is defined to be $\mathsf{Adv}^s(\mathcal{A}) := 2\mathsf{Pr}[\mathsf{S}] - 1$. A PAKE protocol is said to have semantically secure SKs if there exists a non-zero constant $C \in \mathbb{N}$ such that for any PPT $\mathcal{A}$, there exists a negligible (in λ) function negl with the property that $\mathsf{Adv}^s(\mathcal{A}) \leq \frac{Cn_{se}}{|\mathsf{PW}|} + \mathsf{negl}(\lambda)$ where n_{se} is an upper bound on the number of send queries the adversary makes.

Authentication: Let $\mathsf{C2S}$ be the event that "there exists some server instance $\mathcal{S}^j$ that is ready to use a SK, but has not had a correct exchange with a client instance." This event measures the adversary's ability to cause client-to-server authentication to fail, i.e. a server thinks it is talking to a correct client when it is not. Here we simply set $\mathsf{Adv}^{c2s}(\mathcal{A}) := \mathsf{Pr}[\mathsf{C2S}]$, and we say that a PAKE achieves client-to-server authentication if there exists a non-zero C such that for any PPT $\mathcal{A}$, there exists a negligible function negl with the property that $\mathsf{Adv}^{c2s}(\mathcal{A}) \leq \frac{Cn_{se}}{|\mathsf{PW}|} + \mathsf{negl}(\lambda)$. Server-to-client authentication is defined similarly.

Intuitively, the constant C represents the number of passwords that can be ruled out per login attempt. Obviously, $C = 1$ is the optimal bound.

We will return to these definitions with precise terminology in Section 3.

2.2 Differences in accepting, terminating, and partnering

In this section, we illustrate how the notions of "having had a correct exchange", "declaring oneself ready to use a SK", and "refusing to further participate in the protocol", among other things, are formalized in various examples of the literature. These formalizations often vary from paper to paper, usually without much justification.

We begin with some technical terminology which is *mostly* common to all papers, but the interpretation and use of which are what always seem to vary:

– Two instances which "have had a correct exchange" are said to be *partnered*;
– An instance that is "ready to use a session key" is said to have *accepted* or *terminated*;
– The "refusal to further participate in the protocol", oddly, does not seem to have a technical term commonly attached to it.

In the examples that follow, we recapitulate precisely how the afore-mentioned points are dealt with and interpreted. We picked these three examples as they can be considered "landmark" papers: The first [5] introduced BPR-style reasoning to PAKE analysis, the second [3] brought in the RoR model, and the (conference version [23]) of the third [24] showed that PAKE is realizable without random oracles.[2] After each example, we also compile a list of additional papers that emulate (or claim to emulate) the example in question. Recall that ε designates the empty string.

[2] Of course, these are not the only beacons in the field; they are just the most relevant to our work.

In the original model from [5]. This model - the first of its kind - is FtG. ***Accepting:*** At any point in time, an instance $\mathcal{U}^i$ may *accept*. This means that it holds a non-ε SK and has computed a non-ε PID and non-ε SID. ***Terminating:*** At any point in time after having accepted, an instance may *terminate*. This means that it will no longer send, nor expect to receive, any more messages. ***Partnering:*** Two instances $\mathcal{U}^i$ and $\mathcal{V}^j$ are partnered if **1)** one is a client and one is a server, **2)** both instances have accepted, **3)** $pid^i_{\mathcal{U}} = \mathcal{V}$ and $pid^j_{\mathcal{V}} = \mathcal{U}$, **4)** $sid := sid^i_{\mathcal{U}} = sid^j_{\mathcal{V}}$, **5)** $sk^i_{\mathcal{U}} = sk^j_{\mathcal{V}}$, and **6)** no other instance accepts with a SID *sid*.

Apparent intended interpretation: Accepting in this model appears to mean being ready to use the session key, since it is under the condition of having accepted that reveal and test queries can be made. However, accepting is different from terminating in that an instance may wish to accept at one point in time, and yet terminate later. This is to model key confirmation: an instance accepts first - it believes it holds a good session key - but waits for its purported partner to send it a confirmation code to terminate.

Some observations: That the session key can be used before a confirmation code is received is puzzling to us. We believe it may be more logical to have the SK be formally accepted as such at termination time, using this terminology.

There does not appear to be any special term for when an instance refuses to continue in the protocol.

Also, notice that formally, for an instance to be partnered, its partner must be unique. More on this uniqueness is in Paragraph 2.3 and Section 4.

No notion of protocol correctness is defined.

Papers following this approach include [32, 10, 11, 29, 33, 8].

The RoR model paper [3]. This paper's primary topic of investigation is PAKE in the three-party setting[3] (or 3-PAKE, as opposed to the two-party setting, or 2-PAKE), but it contains contributions (namely, the RoR method) relevant to both 3-PAKE and 2-PAKE. (In this paper, we consider only 2-PAKE, which we shall also continue calling just "PAKE".) We focus on the 2-PAKE definitions, referring to the 3-PAKE ones when appropriate. ***Accepting:*** For 2-PAKE, accepting is not formally defined, so it is unclear whether a session key exists at this point or not. For 3-PAKE, accepting only happens "after receiving the last expected protocol message", so this actually corresponds more to termination in [5]. It is unclear whether the 2-PAKE definition is assumed to be the same. ***Terminating:*** This is not formally defined. However, it is stated that "in practice, the SID can be taken to be the partial transcript of the conversation between the client and the server instances before the acceptance." This implies that accepting comes earlier than something else, possibly termination. ***Partnering:*** For 2-PAKE, two instances $\mathcal{U}^i$ and $\mathcal{V}^j$ are partnered if **1)** both instances have accepted, **2)** $sid := sid^i_{\mathcal{U}} = sid^j_{\mathcal{V}}$, **3)** "the partner identifier for $\mathcal{U}^i$ is $\mathcal{V}^j$ and vice-versa", and **4)** "no instance other than $\mathcal{U}^i$ and $\mathcal{V}^j$ accepts with a partner identifier equal to $\mathcal{U}^i$ or $\mathcal{V}^j$".

Observations: Accepting and terminating are unclear. Accepting here may be the same as terminating in [5].

Nothing is in place to indicate if an instance refuses to continue in the protocol.

In terms of partnering, for 2-PAKE there is no need for one instance to be a client and the other a server, unlike in [5]. Points **3)** and **4)** are unclear, because the PID *of an instance* is never defined anywhere. However, these points should be probably be understood respectively as "$pid^i_{\mathcal{U}} = \mathcal{V}$ and $pid^j_{\mathcal{V}} = \mathcal{U}$" and "no other instance accepts with a SID *sid*", as in [5]. Also, in contrast to [5], there is no condition on the SK anymore.

[3] A server aids two clients that wish to exchange a key between themselves; each client shares a private password with the server.

The uniqueness condition is still required to formally satisfy partnering.

There is no mention of correctness.

Papers following this approach include [27, 2, 1].

The journal version [24] of [23]. Paper [23] presented the first practical PAKE secure under standard assumptions (but with a CRS). We discuss the journal version [24] here, which is FtG. ***Accepting:*** For a given instance $\mathcal{U}^i$, once $\mathcal{U}^i$ accepts, $sid^i_{\mathcal{U}}$, $pid^i_{\mathcal{U}}$, and $sk^i_{\mathcal{U}}$ are no longer ε. Also, acceptance implies termination. ***Terminating:*** Terminating means the instance will no longer send nor receive messages. ***Partnering:*** Two instances $\mathcal{U}^i$ and $\mathcal{V}^j$ are partnered if **1)** one is a client and the other a server, **2)** $sid^i_{\mathcal{U}} = sid^j_{\mathcal{V}} \neq \varepsilon$, and **3)** $pid^i_{\mathcal{U}} = \mathcal{V}$ and $pid^j_{\mathcal{V}} = \mathcal{U}$.

Observations: Accepting and terminating are distinct, but unlike in [5], accepting implies termination. This suggests that termination without acceptance is used, in this paper, to designate when an instance refuses to continue.

Partnering differs here from [5] in that acceptance is no longer required, there is no condition on the SK, and there is no requirement of partner uniqueness.

There is a correctness notion: If $\mathcal{U}^i$ and $\mathcal{V}^j$ are partnered, then they must have both accepted and have equal session keys.

The authors state that their definition only covers implicitly authenticated protocols, and that partnering would have to be redefined in order to account for explicit authentication.

Papers following this approach include [17, 22, 19, 25, 26][4].

Having such a variety of definitions to pin down formally fundamental notions is problematic. First, while we have not attempted to show so in this work, it is more than likely that protocols deemed secure according to one definition become trivially insecure according to another definition. Situations like these have been documented in the past in non-password-based cases, see [14, 15]. Secondly, it is confusing for anybody trying to decide whether a protocol's formal security can be trusted or not.

In what follows, we focus on the "uniqueness of partners" aspect in these definitions. We begin by taking another look at the RoR model as defined in [3].

2.3 A bug in the RoR model

In [3], it seems the RoR definition is slightly ill-defined.

Consider the following scenario in the RoR security game. Suppose that the bit flipped at the beginning of the game ends up being 0, making $\mathcal{CH}$ output all-random keys. Suppose also that at some point in time, the adversary $\mathcal{A}$ gets a pair of instances $\mathcal{U}^i$ and $\mathcal{V}^j$ to accept and be partners according to the definition in Section 2.2. In particular, at this point in the game, $\mathcal{U}^i$ and $\mathcal{V}^j$ are the only instances to have accepted with a SID equal to $sid := sid^i_{\mathcal{U}}$. Next, $\mathcal{A}$ performs a `test` query on each instance. The result is that $\mathcal{A}$ receives the same random string R from both instances. Now, suppose that somehow $\mathcal{A}$ gets a third instance $\mathcal{V}^k$ with $pid^k_{\mathcal{V}} = \mathcal{U}$ to accept with SID equal to sid. This removes the uniqueness part of the partnering definition, thus $\mathcal{U}^i$ and $\mathcal{V}^j$ are not partnered anymore. Hence, the result of the `test` query is formally inconsistent with the definition, because $\mathcal{U}^i$ and $\mathcal{V}^j$ should have independent random keys, now that they are not partnered. As for $\mathcal{V}^k$, it is unclear really which key should be assigned in the event a `test` occurs. Should it be random in order to remain consistent with the definition, or should it be set to R in order to be consistent with $\mathcal{A}$'s view?

[4] [19] contains a notion of *semi*-partnering in order to have a definition for instances that have had a correct exchange even if the last message has not been delivered. We adopt this further in this work.

This definitional problem can be solved by changing the requirements of partnering with respect to uniqueness. Note however that ultimately, it is reasonable to expect that double partnering should occur with negligible probability only anyway. However, we cannot make any security statements to enforce this at this point in the model. It must be a proven property, so should not be integrated into the definition.

3 A well-motivated definition

In this section, we build on - and complete - all of the definitions above by supplying a model of our own. We then show how it could fit several protocol formats, covering all typical scenarios of implicit and explicit, unilateral and mutual authentication.

In order to obtain a full description of the formal model, one could simply replace the phrases "ready to use a session key", "refuses to pursue the protocol", and "has had a correct exchange with $\mathcal{U}^i$" with "has accepted", "has aborted", "is partnered to $\mathcal{U}^i$" respectively.

In order to stay with the habits of the literature, we continue to use the random variables for SID, PID, and SK, and the terms "accepting", "terminating", and "partnering". We add explicit terms to indicate when an instance simply stops, and when it refuses to continue mid-protocol.

The notions are compatible with both the FtG and RoR methodologies.

A full recap of the model we propose (taking into account our discussion on uniqueness of partners in Section 4) can be found in Appendix B.

3.1 The definition itself

Status of instances and partnering.

Halting. An instance halts if it stops sending and receiving messages, and ceases to compute anything.

Halting can either be "good" (i.e. with a SK) or "bad" (i.e. without a SK).

Accepting. An instance $\mathcal{U}^i$ accepts if and only if $sid^i_{\mathcal{U}}$ is set to a non-ε value. Accepting means that $\mathcal{U}^i$ believes it is holding enough information to compute a SK. The instance has not necessarily halted.

This formally includes the possibility that it may have actually computed the SK value, but is not yet willing ot use it.

Terminating. An instance $\mathcal{U}^i$ terminates if and only if $sk^i_{\mathcal{U}}$ is set to a non-ε value. If an instance terminates, it halts. Terminating means $\mathcal{U}$ believes it holds a good SK, and is now willing to use it in higher-level applications. $\mathcal{U}^i$ will no longer send nor receive PAKE protocol messages.

If an instance terminates, it accepts, or has previously accepted. In both cases, $sid^i_{\mathcal{U}}$ is set to a non-ε value and remains so. If an instance accepts, it has not necessarily terminated.[5]

Aborting. An instance $\mathcal{U}^i$ aborts if it halts without having terminated.

In other words, it has stopped participating in the protocol exchange, and is unwilling to assign a value to $sk^i_{\mathcal{U}}$. Aborting can very well happen after accepting: Just imagine an instance holding a SK, but waiting for the last confirmation message to finally start using this SK.

[5] We stuck to the idea in [5] that accepting may happen before terminating, even though the term "accepting" seems better suited to designate "successful termination". We did this because the original BPR model is still the most used, so it is probable that this is how the terminology is commonly understood.

Semi-partnering. $\mathcal{U}^i$ and $\mathcal{V}^j$ are semi-partnered if **1)** one is a client and one is a server, **2)** $pid_{\mathcal{U}}^i = \mathcal{V}$ and $pid_{\mathcal{V}}^j = \mathcal{U}$, **3)** $sid_{\mathcal{U}}^i \neq \varepsilon$, $sid_{\mathcal{V}}^j \neq \varepsilon$, and $sid_{\mathcal{U}}^i = sid_{\mathcal{V}}^j$.

Partnering. $\mathcal{U}^i$ and $\mathcal{V}^j$ are partnered if **1)** they are semi-partnered and **2)** $sk_{\mathcal{U}}^i \neq \varepsilon$, $sk_{\mathcal{V}}^j \neq \varepsilon$, and $sk_{\mathcal{U}}^i = sk_{\mathcal{V}}^j$.

By definition, if an instance is semi-partnered to another, it has accepted, and holds a SID. If it is partnered to another, it has terminated and holds a SK.

Explicitly defining semi-partnering and partnering in this way seems to be the only way to have a unique formal treatment of security regardless of whether instances accept and terminate at the same time. In case an instance terminates after accepting, we still want to be able to express at acceptance time that it may have another unique instance to which it is bound, hence the semi-partnering.

To properly define security in the RoR model, and in particular to eliminate the bug identified in Section 2.3, it will be convenient to have the following notion:

Partnering graph. A partnering graph is a graph with instances for nodes. Two nodes have an edge if and only if corresponding instances are partners.

Correctness. Let $\mathcal{U}$ be a client and $\mathcal{V}$ be a server. If $\mathcal{U}^i$ with $pid_{\mathcal{U}}^i = \mathcal{V}$ and $\mathcal{V}^j$ with $pid_{\mathcal{V}}^j = \mathcal{U}$ run the protocol fully and correctly, $\mathcal{U}^i$ and $\mathcal{V}^j$ are partnered.

Remarks: Using the language of graphs to make the definition work may seem to add some rather heavy-handed complexity, but we believe this to be only superficially true. Indeed, in a secure protocol, we want partners to be unique, thus reducing partnering graph sizes to having at most two nodes. Hence, in practice the added complication disappears quite easily.

Correctness can be formulated formally using *matching conversations* [6]. Note that correctness says that matching conversations lead to partnering. However, just because two instances are partnered does not mean that they have had a matching conversation. Intuitively, the protocol should provide a session key that is secure for use once the instances are partnered.

Queries in the FtG and RoR models. The list of queries in Section 2.1 remains exactly the same. We only re-write the test and reveal queries with the technical terms.

We begin by revisiting ***freshness*** for the FtG model: an instance $\mathcal{U}^i$ is said to be *fresh* if it is in a partnering graph in which no instance has been the target of a reveal query.

- (FtG) test$(\mathcal{U}, i)$: If $\mathcal{U}^i$ has not terminated or is not fresh, this returns $\perp$. Otherwise, $\mathcal{CH}$ flips a coin b outside of $\mathcal{A}$'s view. If $b = 0$, a random string R is drawn from the session key space and $tk_{\mathcal{U}}^i \leftarrow R$. Otherwise, $tk_{\mathcal{U}}^i \leftarrow sk_{\mathcal{U}}^i$. $tk_{\mathcal{U}}^i$ is then returned to $\mathcal{A}$. The test query may only be used once in the game.
- (FtG) reveal$(\mathcal{U}, i)$: If $\mathcal{U}^i$ has not terminated or if it is part of a partnering graph in which an instance has been tested, the query returns $\perp$. Otherwise, it returns $sk_{\mathcal{U}}^i$ to $\mathcal{A}$.
- (RoR) test$(\mathcal{U}, i)$: If $\mathcal{U}^i$ has not terminated, $\perp$ is returned. Otherwise, suppose first that $b = 0$. If $\mathcal{U}^i$ is not partnered to any instance, or no instance in the partnering graph it is a part of was subjected to a test query, $\mathcal{CH}$ selects a random R from the session key space and sets $tk_{\mathcal{U}}^i \leftarrow R$. If $\mathcal{U}^i$ is within a partnering graph where some instance $\mathcal{V}^j$ has been tested, $\mathcal{CH}$ sets $tk_{\mathcal{U}}^i \leftarrow tk_{\mathcal{V}}^j$. Suppose now that $b = 1$. In this case, $\mathcal{CH}$ sets $tk_{\mathcal{U}}^i \leftarrow sk_{\mathcal{U}}^i$. Then, $\mathcal{A}$ receives $tk_{\mathcal{U}}^i$.

It should be clear that the introduction of the partnering graph and the absence of any uniqueness requirement in the partnering definition eliminate the bugs raised in the RoR model. Of course, it is merely a tool to keep the definitions consistent; as we have pointed out before, one would want at most one partner to exist. The problem of partner uniqueness is studied in Section 4.

3.2 Examples of how it functions

We give here a few examples of protocol structures that fit our definition in various ways. In what follows, m and μ basically represent the main protocol flows, i.e. the messages that a shared secret is usually computed with. The symbols sid and sk designate the SID and SK. The k and κ values are *confirmation codes*; their role is to prove to the other party that the same shared secret was computed at both ends of the protocol run. As such, they must be computed from, or at the same time as, the shared secret.

We look at two-pass, three-pass, and four-pass protocols, but one can easily construct similar examples with protocols having more messages.

In practice, the SID is usually taken to be the concatenation of $\mathcal{C}$, $\mathcal{S}$, m, and μ. This makes the randomness of both parties an input to SID. We shall return to this point later.

Acceptance and termination occur in one step for both the client and the server. Examples of protocols like this are OMDHKE in [11] and EKE2 in [5]. The former achieves *explicit authentication of the server to the client*, as in Figure 1 and the latter achieves *implicit authentication.* (Take Figure 1 and remove all mention of κ and k.) Obviously, in two-pass protocols the parties involved have no other choice but to accept and terminate at the same time. Also, it is clear that the receiver of the first protocol message cannot be assured that it is talking to a live instance.

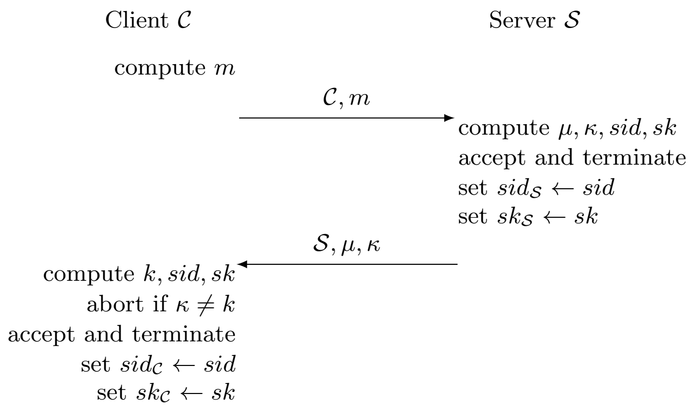
Client $\mathcal{C}$
Server $\mathcal{S}$
compute m
$\mathcal{C}, m$
compute μ, κ, sid, sk
accept and terminate
set $sid_{\mathcal{S}} \leftarrow sid$
set $sk_{\mathcal{S}} \leftarrow sk$
$\mathcal{S}, \mu, \kappa$
compute k, sid, sk
abort if $\kappa \neq k$
accept and terminate
set $sid_{\mathcal{C}} \leftarrow sid$
set $sk_{\mathcal{C}} \leftarrow sk$

Fig. 1.

One party accepts and terminates in one step, the other accepts first and terminates later. In this class of protocols, we find e.g. the protocol of Groce and Katz [19], the OEKE protocol from [10], and the F + PaKE protocol from [17]. The Groce-Katz protocol achieves mutual authentication, as in Figure 2. OEKE and F + PaKE only achieve explicit authentication of the client to the server. (Remove all mention of κ_1

and k_1 in Figure 2.) In both cases, acceptance by the server occurs right before sending the second message, and termination occurs at the very end.

After $\mathcal{C}$ sends the last message, but before this message is received by $\mathcal{S}$, $\mathcal{C}$ and $\mathcal{S}$ are semi-partnered.

In this class of protocols, the code κ_2 and SK sk need not be computed by the server once it receives the first message (as shown in Figure 2); this can be postponed to the end.[6]

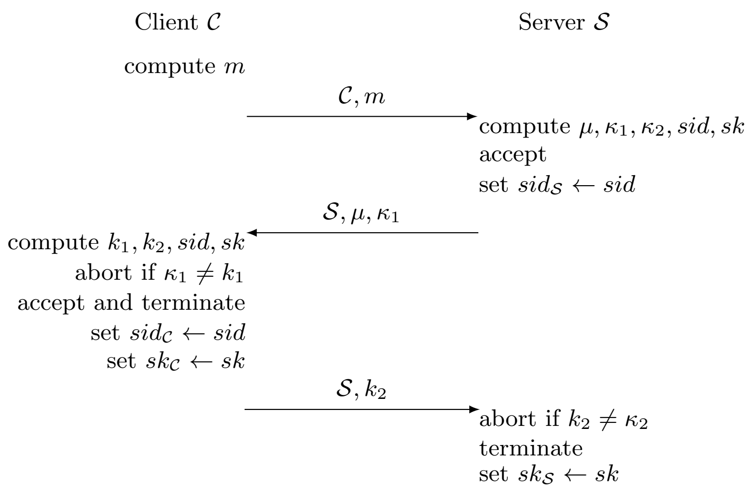
Client $\mathcal{C}$
Server $\mathcal{S}$
compute m
$\mathcal{C}, m$
compute $\mu, \kappa_1, \kappa_2, sid, sk$
accept
set $sid_{\mathcal{S}} \leftarrow sid$
$\mathcal{S}, \mu, \kappa_1$
compute k_1, k_2, sid, sk
abort if $\kappa_1 \neq k_1$
accept and terminate
set $sid_{\mathcal{C}} \leftarrow sid$
set $sk_{\mathcal{C}} \leftarrow sk$
$\mathcal{S}, k_2$
abort if $k_2 \neq \kappa_2$
terminate
set $sk_{\mathcal{S}} \leftarrow sk$

Fig. 2.

Both parties accept and terminate in two steps. This happens for instance in the AMP protocol from [28], see Figure 3. Both client and server accept at different stages. Note that sometimes protocols of this form can regroup certain messages in order to yield n-pass rather than $(n+1)$-pass protocols. (However, four-pass protocols have some advantages over three-pass ones too, e.g. in [29] Kwon reports on a practical *multiple-password* online guessing attack against three-pass protocols, that four-pass protocols do not suffer from. This attack depends on the server's *actual response time* in processing an authentication request, so falls out of the scope of currently used security models.)

As in the previous case, the actual computation of the SK (which is different from it being formally accepted as usable) can be moved around somewhat, this time both at the server and client end.

4 The quality of partner uniqueness

In this section, we examine just exactly "how unique" partners can be in AKEs, with a special treatment of PAKEs, ***assuming we stick to the definitions of partnering and testing that do not take into account our modifications, in particular with no notion of partnering graph. We also assume a partnering definition that does not mention uniqueness, e.g. that of [17, 22, 19, 25, 26].*** This serves

[6] This may even be desirable for efficiency reasons.

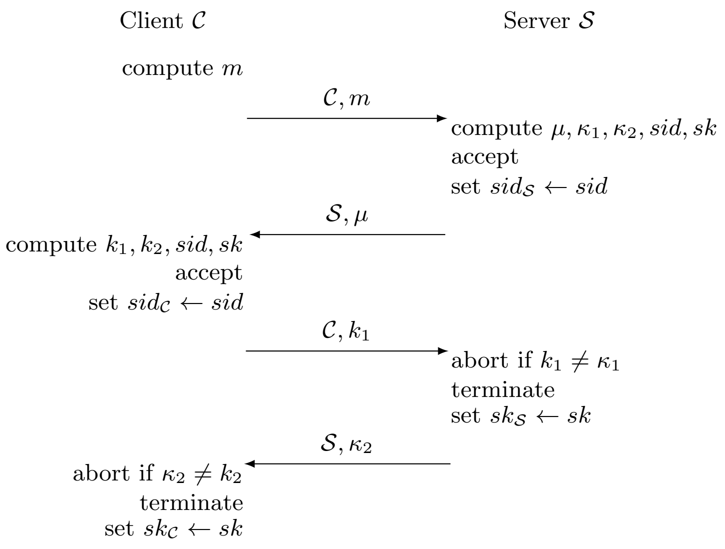
Client $\mathcal{C}$
Server $\mathcal{S}$
compute m
$\mathcal{C}, m$
compute $\mu, \kappa_1, \kappa_2, sid, sk$
accept
set $sid_{\mathcal{S}} \leftarrow sid$
$\mathcal{S}, \mu$
compute k_1, k_2, sid, sk
accept
set $sid_{\mathcal{C}} \leftarrow sid$
$\mathcal{C}, k_1$
abort if $k_1 \neq \kappa_1$
terminate
set $sk_{\mathcal{S}} \leftarrow sk$
$\mathcal{S}, \kappa_2$
abort if $\kappa_2 \neq k_2$
terminate
set $sk_{\mathcal{C}} \leftarrow sk$

Fig. 3.

as further evidence that partnering and partner uniqueness must be carefully treated, and probably separately.

In general, we note that it seems uniqueness of partners has completely disappeared from AKE requirements in a very large proportion of AKE research papers. Also, when uniqueness is mentioned, it is not even clear how probable this uniqueness should be (e.g. [5, 3, 26]). This is odd, since partner uniqueness was actually considered in perhaps the very first AKE modeling paper [6]. Of course, it is quite natural to expect that for *all* AKEs, if an instance runs the protocol it should have at most *one* partner with *overwhelming* probability, but this is rarely explicitly mentioned. One reason for this may be that usually an attempt is made at funneling all of the desirable security properties into the definition of session key security, but there are two problems with this, which we show below.

First, even in the general case not all security properties can be treated through this mechanism, and it so happens that uniqueness of partners is one of them. Secondly, in the PAKE case - where the optimal bound for semantic security is not even a negligible value - the property may formally fail altogether.

4.1 An obstacle caused by the test query

We begin with some flawed reasoning that applies to all AKE settings, and concerning the link between multiple partnering of instances and SK security. Again, we stress that this is specific to ***a partnering definition without partnering graphs, and without built-in uniqueness.*** Also, we use ***the RoR setting as an example.***

What we cannot do... Let AKE be an authenticated key exchange protocol. In the game played in Section 2.1, we can always consider the event "there exist distinct users $\mathcal{U}$ and $\mathcal{V}$, and distinct instances $\mathcal{U}^i$, $\mathcal{V}^j$, and $\mathcal{V}^k$ such that $\mathcal{U}^i$ and $\mathcal{V}^j$ are partnered and $\mathcal{U}^i$ and $\mathcal{V}^k$ are partnered", which we denote MP (for Multiple Partnering). For any adversary $\mathcal{B}$, let $\mathsf{Adv}^{mp}(\mathcal{B}) := \mathsf{Pr}[\mathsf{MP}]$. Our objective is to relate $\mathsf{Adv}^{mp}(\mathcal{B})$ to $\mathsf{Adv}^{s}(\mathcal{A})$

for a suitably constructed $\mathcal{A}$. Ultimately, it would be nice if the negligibility of the latter implied that of the former.

• ***Construction*** $\mathfrak{C}$***:*** *Fix some adversary* $\mathcal{B}$*, and consider adversary* $\mathcal{A}$*, trying to break the semantic security of* AKE*, designed as follows.* $\mathcal{A}$ *runs exactly like* $\mathcal{B}$*, but examining whether or not* $\mathcal{B}$ *can cause* MP *to occur. Whenever an instance terminates,* $\mathcal{A}$ *checks to see if* MP *has happened. If* $\mathcal{B}$ *halts and* MP *never happened,* $\mathcal{A}$ *flips a coin* b' *and outputs* b'*. As soon as* MP *happens (if it does),* $\mathcal{A}$ *stops* $\mathcal{B}$*, and studies the involved instances. Let* $\mathcal{U}^i$*,* $\mathcal{V}^j$*, and* $\mathcal{V}^k$ *be these instances.* $\mathcal{A}$ *performs one* test *query on* $\mathcal{V}^j$ *and another on* $\mathcal{V}^k$*, receiving* $tk^j_{\mathcal{V}}$ *and* $tk^k_{\mathcal{V}}$*. If* $tk^j_{\mathcal{V}} = tk^k_{\mathcal{V}}$*,* $\mathcal{A}$ *sets* $b' \leftarrow 1$*, and otherwise* $b' \leftarrow 0$*.* $\mathcal{A}$ *then outputs* b' *and halts.*

At this point, in the event that MP does indeed occur, it is tempting to directly conclude that since the instances $\mathcal{V}^j$ and $\mathcal{V}^k$ are clearly not partners (since their PIDs do not correspond) but do hold identical session keys (since they are each partnered to $\mathcal{U}^i$), the test queries performed should give either independent random keys if $b = 0$ or identical keys if $b = 1$. From this, one immediately sees that if semantic security of the session key holds, then MP may only occur with negligible probability as well. However, we cannot make this assertion, because it may be that $\mathcal{U}^i$ has also had a test query performed on it. It the RoR model, this would make all keys identical no matter the value of b.[7]

Unfortunately, there does not seem to be any way around this obstruction. It looks as though one would have to somehow need the probability that $\mathcal{A}$ makes a test query on $\mathcal{U}^i$ to be itself a negligible value, but this is not justifiable. Thus, in order to prove anything in this way, one has to consider a more restricted version of MP.

...what we can do... We consider event MP^* defined as "there exist distinct users $\mathcal{U}$ and $\mathcal{V}$, and distinct instances $\mathcal{U}^i$, $\mathcal{V}^j$, and $\mathcal{V}^k$, such that **1)** $\mathcal{U}^i$ and $\mathcal{V}^j$ are partnered, **2)** $\mathcal{U}^i$ and $\mathcal{V}^k$ are partnered, and **3)** no test query was performed on $\mathcal{U}^i$."

We also consider construction $\mathfrak{C}^*$ which is basically identical to construction $\mathfrak{C}$, except that $\mathcal{A}$ looks out for event MP^* rather than MP. The next lemma precisely relates $\mathsf{Adv}^{mp^*}(\mathcal{B})$ to $\mathsf{Adv}^{s}(\mathcal{A})$.

Lemma 1 *It holds that* $\mathsf{Adv}^{s}(\mathcal{A}) = \big(1 - \frac{1}{2^{\ell-1}}\big)\mathsf{Adv}^{mp^*}(\mathcal{B})$*.*

Proof: We have

$$\mathsf{Pr}[S] = \mathsf{P}[\mathsf{S}|\mathsf{MP}^*]\mathsf{Pr}[\mathsf{MP}^*] + \mathsf{P}[\mathsf{S}|\neg\mathsf{MP}^*]\mathsf{Pr}[\neg\mathsf{MP}^*] \tag{1}$$

Conditioned on MP^* not having occurred, by definition of $\mathcal{A}$ we have $\mathsf{P}[\mathsf{S}|\neg\mathsf{MP}^*] = \frac{1}{2}$. If MP^* has occurred, all three instances should hold the same SK sk. However, since $pid^j_{\mathcal{V}} = pid^k_{\mathcal{V}} = \mathcal{U}$, $\mathcal{V}^j$ and $\mathcal{V}^k$ are not partnered. Finally, no test query was performed on $\mathcal{U}^i$. Therefore, if $b = 0$ the $\mathsf{test}(\mathcal{V}, j)$ and $\mathsf{test}(\mathcal{V}, k)$ queries output $tk^j_{\mathcal{V}}$ and $tk^k_{\mathcal{V}}$ independently and randomly. Thus, $\mathcal{A}$ will output the correct bit value unless these keys collide, which may happen with probability $\frac{1}{2^\ell}$. Plugging these values into Equation 1 gives $\mathsf{Pr}[S] = \frac{1}{2}\big(1 - \frac{1}{2^{\ell-1}}\big)\mathsf{Pr}[\mathsf{MP}] + \frac{1}{2}$. Taking the advantage function formulas finishes the proof. ■

On one hand, this lemma shows that for AKE protocols for which Adv^s is required to be negligible (such as PKI-AKEs, or SSK-AKEs), the probability that MP^* occurs

[7] Similar reasoning shows that the FtG model suffers from the phenomenon as well, basically because if $\mathcal{U}^i$ is tested, the freshness condition prohibits testing of the two other instances. Thus, our observation is valid "beyond RoR".

is negligible as well. On the other hand, for PAKEs, this no longer holds: All we can say is

$$\mathsf{Adv}^{mp^*}(\mathcal{B}) \leq \left(\frac{2^{\ell-1}}{2^{\ell-1}-1}\right)\frac{Cn_{se}}{|\mathsf{PW}|} \tag{2}$$

where C and n_{se} are as in Paragraph 2.1.

...and what we can assert. Hence, we can basically gather the following three points:

- **1)** Adv^s's negligibility does not immediately imply that event MP will occur with negligible probability. Notably, this concerns the PKI and SSK cases.
- **2)** Adv^s's negligibility only implies that event MP^*will occur with negligible probability. This is certainly a desirable guarantee, but remains weaker than MP.
- **3)** In the PAKE case, even the restricted event MP^* has no immediate reason to be negligible at all.

Point **3)** is further illustrated in Paragraph 4.2 below, where we propose a PAKE that is secure according the the model of Paragraph 2.1, but at the same time can cause MP^* to occur with a probability nearly optimally respecting the bound in Equation 2.

4.2 A "secure" PAKE protocol where non-negligible multiple partnering may occur

We now describe our flawed protocol, dubbed P. It actually is not a "pure" PAKE, since the client authenticates the server using a public key, while the server authenticates the client using a password. We were unable to construct a suitable example in the "pure" setting, but we believe our point is still valid. (However, a "pure" example would be very interesting.)

Setup. From here on, $(\mathsf{SKeyGen}, \mathsf{Sig}, \mathsf{Ver})$ is a strongly-EU-CMA-secure (see [4])[8] signature scheme and $(\mathsf{EKeyGen}, \mathsf{Enc}, \mathsf{Dec})$ is a CCA-2-secure public key encryption scheme. We assume that there can be many different client identities, but only one server identity $\mathcal{S}$. Thus, there are many clients, and many client instances, but only one server, and many server instances. The KeyGen algorithms are run once to obtain public key/secret key pairs (pk_E, sk_E) and (pk_S, sk_S) for encryption and signing respectively. Each client $\mathcal{C}$ has its password $pw_{\mathcal{C}}$ registered at server $\mathcal{S}$, and has (pk_E, pk_S), say, hardcoded into its software specification. Server $\mathcal{S}$ holds the full password file $\{pw_{\mathcal{C}}\}_{\mathcal{C}}$, as well as (sk_E, sk_S).[9]

Running P. The protocol flows are shown in Figure 4. First, the server $\mathcal{S}$ pings the client $\mathcal{C}$ with a signed nonce. $\mathcal{C}$ then verifies the signature, accepts without terminating, and returns an encryption of the nonce it just received, a fresh nonce of its own, a session key it selects itself randomly, and its password. Upon receiving this encryption, $\mathcal{S}$ decrypts and checks if the password matches $\mathcal{C}$'s identity and if it recognizes its own nonce. If so, it accepts and terminates, validates the session key, and returns to $\mathcal{C}$ a new signature on both nonces. Finally, if the signature verifies $\mathcal{C}$ terminates and validates the session key.

Upon acceptance, for our protocol the SID is set as being $(M, \mathcal{C}, \mathcal{S})$, where M is $\mathcal{C}$'s chosen nonce. As the reader may expect, the fact that the SID is a function of *only the client's nonce* is what causes trouble.

[8] The fact that signatures are *strongly* secure is used to make the security proof simpler, but is not strictly necessary.

[9] One may think of a setup of this sort as being implemented e.g. for a large group of employees in a company.

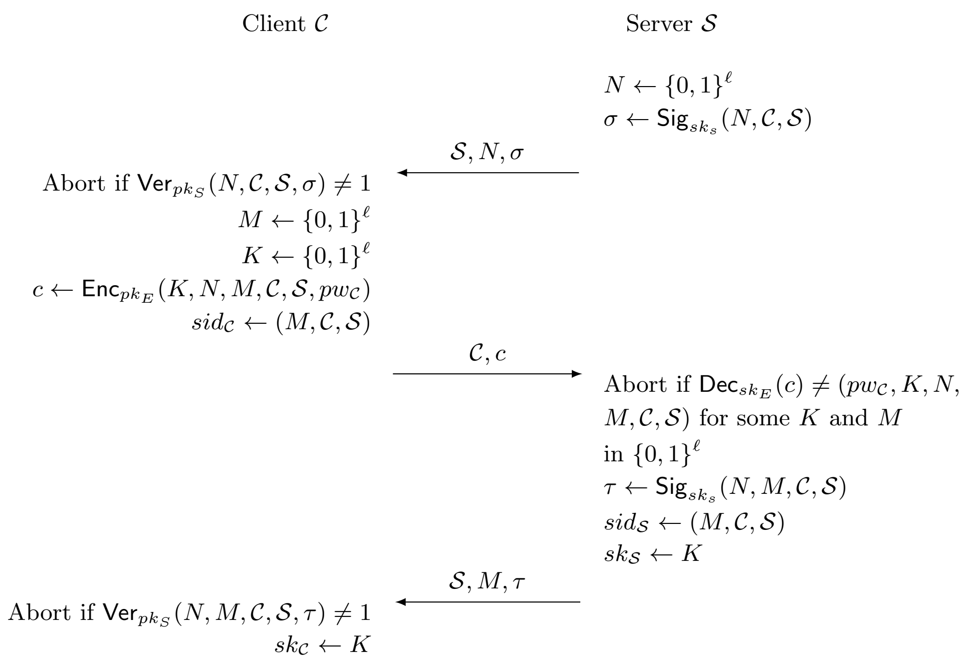
Client $\mathcal{C}$
Server $\mathcal{S}$
$N \leftarrow \{0,1\}^{\ell}$
$\sigma \leftarrow \mathsf{Sig}_{sk_s}(N, \mathcal{C}, \mathcal{S})$
$\mathcal{S}, N, \sigma$
Abort if $\mathsf{Ver}_{pk_S}(N, \mathcal{C}, \mathcal{S}, \sigma) \neq 1$
$M \leftarrow \{0,1\}^{\ell}$
$K \leftarrow \{0,1\}^{\ell}$
$c \leftarrow \mathsf{Enc}_{pk_E}(K, N, M, \mathcal{C}, \mathcal{S}, pw_{\mathcal{C}})$
$sid_{\mathcal{C}} \leftarrow (M, \mathcal{C}, \mathcal{S})$
$\mathcal{C}, c$
Abort if $\mathsf{Dec}_{sk_E}(c) \neq (pw_{\mathcal{C}}, K, N, M, \mathcal{C}, \mathcal{S})$ for some K and M in $\{0,1\}^{\ell}$
$\tau \leftarrow \mathsf{Sig}_{sk_s}(N, M, \mathcal{C}, \mathcal{S})$
$sid_{\mathcal{S}} \leftarrow (M, \mathcal{C}, \mathcal{S})$
$sk_{\mathcal{S}} \leftarrow K$
$\mathcal{S}, M, \tau$
Abort if $\mathsf{Ver}_{pk_S}(N, M, \mathcal{C}, \mathcal{S}, \tau) \neq 1$
$sk_{\mathcal{C}} \leftarrow K$

Fig. 4. The P protocol.

Discussion. On one hand, P definitely looks like a bad protocol, given that it is absolutely littered with red flags:

- **1)** The SID depends only on the random bits of one of the two parties involved;
- **2)** The session key is completely determined by one of the two parties involved;
- **3)** The session key and identifier are completely decoupled.

These points certainly go against well-established design principles that PAKEs commonly follow. As for item **4)**, it illustrates the need to study uniqueness of partners as a security property in its own right in the PAKE case.

On the other hand, P actually satisfies an appropriately modified BPR-style definition of security for PAKEs:

Theorem 1 *Protocol* P *is a secure PAKE in that for any efficient adversary* $\mathcal{A}$*, when* $\mathcal{A}$ *plays the RoR game against challenger* $\mathcal{CH}$ *as described in Section 2.1, we have* $\mathsf{Adv}^s(\mathcal{A}) \leq \frac{n_{se}}{|\mathsf{PW}|} + \mathsf{negl}(\lambda)$*, where* n_{se} *is an upper bound on the number of send queries made by* $\mathcal{A}$*.*

Proof: See Appendix A. ■

Furthermore, P also suffers from point **4)** below:

- **4)** Event MP^* may occur with non-negligible probability.

A demonstration of this is in the next paragraph.

Partnering a client instance to two server instances. We construct a specific attacker $\mathcal{B}$ in the security model described in Section 2.1. $\mathcal{B}$ is trying to cause the event MP^*.

$\mathcal{B}$ first initializes a client instance $\mathcal{C}^1$ and server instance $\mathcal{S}^1$ with $pid_{\mathcal{C}}^1 = \mathcal{S}$ and $pid_{\mathcal{S}}^1 = \mathcal{C}$. Next, it performs an $\mathsf{execute}$ query on these instances. They now share a SK sk and a SID $(M, \mathcal{C}, \mathcal{S})$. Then, it performs a test query on $\mathcal{S}^1$ to get sk. Now, for $j = 2, \ldots$ it repeats the following steps until some server instance $\mathcal{S}^j$ accepts:

– **1)** It initializes $\mathcal{S}^j$ with $pid_{\mathcal{S}}^j = \mathcal{C}$, and instructs it to send the first protocol message $\mathcal{S}, N^j, \sigma$;
– **2)** It chooses $pw^j \leftarrow \mathsf{PW}$ randomly and computes $c \leftarrow \mathsf{Enc}_{pk_E}(sk, N^j, M, \mathcal{C}, \mathcal{S}, pw^j)$;
– **3)** It sends c^j to $\mathcal{S}^j$ and observes whether $\mathcal{S}^j$ accepts or not.

When some $\mathcal{S}^j$ finally accepts, $\mathcal{B}$ has guessed the right password $pw_{\mathcal{C}}$, and the instances $\mathcal{C}^1$ and $\mathcal{S}^j$ are *partnered*: $sid_{\mathcal{S}}^j = sid_{\mathcal{C}}^1 = (M, \mathcal{C}, \mathcal{S})$ and $sk_{\mathcal{S}}^j = sk_{\mathcal{C}}^1 = sk$.

In the above scenario, it should be clear that if t is the number of $\mathcal{S}^j$ instances (for $j \geq 2$) used in order to succeed, we have

$$\mathsf{Adv}^{mp^*}(\mathcal{B}) = \mathsf{Pr}[\mathsf{MP}^*] = \frac{t}{|\mathsf{PW}|}$$

which is certainly not negligible. Also, since we have $t \leq n_{se} - 2$, this is well in accordance with Equation 2 (taking $C = 1$, the best possible constant), and thus with the required bound on semantic security.

4.3 Lessons learned on requirements

We should conclude that MP has to be considered as an event to render negligible in its own right for both AKEs, where the semantic security of SK only provides a safety net in that it renders a more restricted event negligible, and PAKEs, where there is no safety net at all. Fortunately, most AKEs in the literature do not have these problems because of the way partnering is instantiated by concrete protocols. In particular, as previously stated most PAKEs build the SIDs from concatenations of almost all messages, so all parties' random values are involved, and uniqueness of SIDs, and therefore of partners, is a trivially verified matter. However, as we have shown, it can formally fail in "non-concatenation" cases, and so it is better off being a stated and formally proven requirement, no matter how trivial. Thus it seems worthwhile to add to the security model the property that MP should be negligible *all the time*. In fact, using our language from Section 3, multiple semi-partnering should be negligible all the time.

In the particular case where SIDs are used to establish partnering - this is almost always the case in BPR-style models - The simplest way to do this would be to prove that the event SID is negligible, where we define SID to be "there exist more than two instances that share the same non$-\varepsilon$ SID". In other words, we add the following point to our model:

Partner uniqueness: Let SID be the event that "there exist more than two instances that have the same non-ε SID". Let $\mathsf{Adv}^{sid}(\mathcal{A}) := \mathsf{Pr}[\mathsf{SID}]$. We say that PAKE achieves unique partnering if for any PPT $\mathcal{A}$ the function Adv^{sid} is negligible. Note that the quality of the long-term keying material here no longer should have an influence on the security bound we require.

For a complete model description, we refer to Appendix B.

5 Conclusion and future work

In this paper, we have shown that there are multiple BPR-style definitions of security for PAKEs in the literature, and have attempted to unify them. In the process, we found a way to solve a bug in the model of [3]. Finally, we showed that uniqueness of partners is a property worth establishing explicitly, similarly to explicit authentication and semantic security of the session key. Specifically, it should hold with overwhelming probability even in the PAKE case, where the other main security properties can only be ensured with non-negligible probability.

As far as this study goes, it should be extended to include long-term key corruptions. Also, it would be extremely interesting to find a counter-example similar to protocol P in the "pure PAKE" setting. Finally, it may be worthwhile to further refine BPR-style models by adding a specific variable to designate the shared secret computed by protocol participants. It seems reasonable that this variable would be non-ε when semi-partnering occurs.

Acknowledgments. We would like to thank the reviewers for their comments. The author is supported by the *Fonds National de la Recherche, Luxembourg*, via the CORE project AToMS and the INTER project SEQUOIA.

References

1. Abdalla, M., Benhamouda, F., MacKenzie, P.: Security of the J-PAKE Password-Authenticated Key Exchange Protocol. In: 2015 IEEE Symposium on Security and Privacy (2015)
2. Abdalla, M., Benhamouda, F., Pointcheval, D.: Public-key encryption indistinguishable under plaintext-checkable attacks. In: Katz, J. (ed.) Public-Key Cryptography – PKC 2015, Lecture Notes in Computer Science, vol. 9020, pp. 332–352. Springer Berlin Heidelberg (2015), `http://dx.doi.org/10.1007/978-3-662-46447-2_15`
3. Abdalla, M., Fouque, P.A., Pointcheval, D.: Password-based authenticated key exchange in the three-party setting. In: Vaudenay, S. (ed.) Public Key Cryptography - PKC 2005, Lecture Notes in Computer Science, vol. 3386, pp. 65–84. Springer Berlin Heidelberg (2005), `http://dx.doi.org/10.1007/978-3-540-30580-4_6`
4. An, J.H., Dodis, Y., Rabin, T.: On the security of joint signature and encryption. In: Proceedings of the International Conference on the Theory and Applications of Cryptographic Techniques: Advances in Cryptology. pp. 83–107. EUROCRYPT '02, Springer-Verlag, London, UK, UK (2002), `http://dl.acm.org/citation.cfm?id=647087.715701`
5. Bellare, M., Pointcheval, D., Rogaway, P.: Authenticated Key Exchange Secure Against Dictionary Attacks. In: Preneel, B. (ed.) Advances in Cryptology – EUROCRYPT 2000. LNCS, vol. 1807, pp. 139–155. Springer (2000)
6. Bellare, M., Rogaway, P.: Entity Authentication and Key Distribution. In: Stinson, D.R. (ed.) Advances in Cryptology – CRYPTO '93. LNCS, vol. 773, pp. 232–249. Springer (1993)
7. Bellovin, S.M., Merritt, M.: Encrypted Key Exchange: Password-Based Protocols Secure Against Dictionary Attacks. In: 1992 IEEE Computer Society Symposium on Research in Security and Privacy, May 4-6, 1992. pp. 72–84 (1992)
8. Benhamouda, F., Blazy, O., Chevalier, C., Pointcheval, D., Vergnaud, D.: New techniques for sphfs and efficient one-round pake protocols. In: Canetti, R., Garay, J. (eds.) Advances in Cryptology CRYPTO 2013, Lecture Notes in Computer Science, vol. 8042, pp. 449–475. Springer Berlin Heidelberg (2013), `http://dx.doi.org/10.1007/978-3-642-40041-4_25`
9. Boyko, V., MacKenzie, P.D., Patel, S.: Provably Secure Password-Authenticated Key Exchange Using Diffie-Hellman. In: Preneel, B. (ed.) Advances in Cryptology – EUROCRYPT 2000. LNCS, vol. 1807, pp. 156–171. Springer (2000)

10. Bresson, E., Chevassut, O., Pointcheval, D.: Security Proofs for an Efficient Password-based Key Exchange. In: Jajodia, S., Atluri, V., Jaeger, T. (eds.) ACM Conference on Computer and Communications Security. pp. 241–250. ACM (2003)
11. Bresson, E., Chevassut, O., Pointcheval, D.: New Security Results on Encrypted Key Exchange. In: Bao, F., Deng, R.H., Zhou, J. (eds.) Public Key Cryptography. LNCS, vol. 2947, pp. 145–158. Springer (2004)
12. Canetti, R.: Universally composable security: A new paradigm for cryptographic protocols. In: Proceedings of the 42Nd IEEE Symposium on Foundations of Computer Science. pp. 136–. FOCS '01, IEEE Computer Society, Washington, DC, USA (2001), http://dl.acm.org/citation.cfm?id=874063.875553
13. Canetti, R., Halevi, S., Katz, J., Lindell, Y., MacKenzie, P.D.: Universally Composable Password-Based Key Exchange. In: Cramer, R. (ed.) Advances in Cryptology – EUROCRYPT 2005. LNCS, vol. 3494, pp. 404–421. Springer (2005)
14. Choo, K.K., Boyd, C., Hitchcock, Y.: Examining indistinguishability-based proof models for key establishment protocols. In: Roy, B. (ed.) Advances in Cryptology - ASIACRYPT 2005, Lecture Notes in Computer Science, vol. 3788, pp. 585–604. Springer Berlin Heidelberg (2005), http://dx.doi.org/10.1007/11593447_32
15. Cremers, C.: Examining indistinguishability-based security models for key exchange protocols: The case of ck, ck-hmqv, and eck. In: Proceedings of the 6th ACM Symposium on Information, Computer and Communications Security. pp. 80–91. ASIACCS '11, ACM, New York, NY, USA (2011), http://doi.acm.org/10.1145/1966913.1966925
16. Diffie, W., Hellman, M.: New directions in cryptography. IEEE Trans. Inf. Theor. 22(6), 644–654 (Sep 2006), http://dx.doi.org/10.1109/TIT.1976.1055638
17. Gennaro, R., Lindell, Y.: A framework for password-based authenticated key exchange. In: Biham, E. (ed.) Advances in Cryptology EUROCRYPT 2003, Lecture Notes in Computer Science, vol. 2656, pp. 524–543. Springer Berlin Heidelberg (2003), http://dx.doi.org/10.1007/3-540-39200-9_33
18. Goldreich, O., Lindell, Y.: Session-key generation using human passwords only. In: Kilian, J. (ed.) Advances in Cryptology CRYPTO 2001, Lecture Notes in Computer Science, vol. 2139, pp. 408–432. Springer Berlin Heidelberg (2001), http://dx.doi.org/10.1007/3-540-44647-8_24
19. Groce, A., Katz, J.: A new framework for efficient password-based authenticated key exchange. In: Proceedings of the 17th ACM Conference on Computer and Communications Security. pp. 516–525. CCS '10, ACM, New York, NY, USA (2010), http://doi.acm.org/10.1145/1866307.1866365
20. Halevi, S., Krawczyk, H.: Public-key cryptography and password protocols. ACM Trans. Inf. Syst. Secur. 2(3), 230–268 (Aug 1999), http://doi.acm.org/10.1145/322510.322514
21. Jablon, D.P.: Strong Password-Only Authenticated Key Exchange. ACM SIGCOMM Computer Communication Review 26(5), 5–26 (1996)
22. Jiang, S., Gong, G.: Password based key exchange with mutual authentication. In: Handschuh, H., Hasan, M. (eds.) Selected Areas in Cryptography, Lecture Notes in Computer Science, vol. 3357, pp. 267–279. Springer Berlin Heidelberg (2005), http://dx.doi.org/10.1007/978-3-540-30564-4_19
23. Katz, J., Ostrovsky, R., Yung, M.: Efficient Password-Authenticated Key Exchange Using Human-Memorable Passwords. In: Pfitzmann, B. (ed.) Advances in Cryptology – EUROCRYPT 2001. LNCS, vol. 2045, pp. 475–494. Springer (2001)
24. Katz, J., Ostrovsky, R., Yung, M.: Efficient and secure authenticated key exchange using weak passwords. JOURNAL OF THE ACM 57(1) (2009)
25. Katz, J., Vaikuntanathan, V.: Smooth projective hashing and password-based authenticated key exchange from lattices. In: Matsui, M. (ed.) Advances in Cryptology ASIACRYPT 2009, Lecture Notes in Computer Science, vol. 5912, pp. 636–652. Springer Berlin Heidelberg (2009), http://dx.doi.org/10.1007/978-3-642-10366-7_37
26. Katz, J., Vaikuntanathan, V.: Round-optimal password-based authenticated key exchange. In: Ishai, Y. (ed.) Theory of Cryptography, Lecture Notes in Computer Science, vol. 6597, pp. 293–310. Springer Berlin Heidelberg (2011), http://dx.doi.org/10.1007/978-3-642-19571-6_18

27. Kiefer, F., Manulis, M.: Oblivious pake: Efficient handling of password trials. Cryptology ePrint Archive, Report 2013/127 (2013), `http://eprint.iacr.org/`
28. Kwon, T.: Authentication and key agreement via memorable password. In: ISOC Network and Distributed System Security Symposium (2001)
29. Kwon, T.: Practical Authenticated Key Agreement Using Passwords. In: Zhang, K., Zheng, Y. (eds.) Information Security, LNCS, vol. 3225, pp. 1–12. Springer Berlin Heidelberg (2004)
30. Lancrenon, J.: On password-authenticated key exchange security modeling. In: PASSWORDS 2015, Cambridge, UK, December 7-9, 2015, Proceedings. LNCS, vol. 9551, pp. 120–143. Springer International Publishing (2016)
31. Lucks, S.: Open key exchange: How to defeat dictionary attacks without encrypting public keys. In: Proceedings of the 5th International Workshop on Security Protocols. pp. 79–90. Springer-Verlag, London, UK, UK (1998), `http://dl.acm.org/citation.cfm?id=647215.720526`
32. MacKenzie, P.: The PAK Suite: Protocols for Password-Authenticated Key Exchange. DIMACS Technical Report 2002-46 (2002), (Page 7)
33. MacKenzie, P., Patel, S., Swaminathan, R.: Password-authenticated key exchange based on rsa. Int. J. Inf. Secur. 9(6), 387–410 (Dec 2010), `http://dx.doi.org/10.1007/s10207-010-0120-3`
34. Pointcheval, D.: Password-Based Authenticated Key Exchange. In: Fischlin, M., Buchmann, J., Manulis, M. (eds.) Public Key Cryptography PKC 2012, LNCS, vol. 7293, pp. 390–397. Springer (2012)
35. Shoup, V.: On Formal Models for Secure Key Exchange. Cryptology ePrint Archive, Report 1999/012 (1999), `http://eprint.iacr.org/1999/012`

A Proof of Theorem 1

Here is the proof of Theorem 1. We actually prove a bit more, in that we also show that P satisfies Client-to-Server (C2S) authentication.

The proof proceeds in a sequence of games. Pr_i and Adv_i designate the probability and advantage functions respectively in the probability space defined by game i.

Game 0. This is the original attack game described in Section 2.

Game 1. If any of the instances' chosen nonces - i.e. the N and M values - collide, abort the game. The probability of this occurring is controlled by the birthday bound $\frac{(n_{se}+n_{ex})^2}{2^{\ell+1}}$ which is negligible. Here, n_{ex} is a bound on the the number of $\mathsf{execute}$ queries $\mathcal{A}$ may make. Thus:

$$\left|\mathsf{Adv}_0^s(\mathcal{A}) - \mathsf{Adv}_1^s(\mathcal{A})\right| \leq \mathsf{negl}(\lambda) \text{ and } \left|\mathsf{Adv}_0^{c2s}(\mathcal{A}) - \mathsf{Adv}_1^{c2s}(\mathcal{A})\right| \leq \mathsf{negl}(\lambda)$$

From this point on, existing client and server instances are uniquely determined by their random nonces.

Game 2. Suppose a client instance $\mathcal{C}^i$ receives the first or third protocol message and the signature verification passes. If said message was not computed by an appropriate server instance, abort the game. The probability of this occurring is controlled by the security of the signature scheme:

$$\left|\mathsf{Adv}_1^s(\mathcal{A}) - \mathsf{Adv}_2^s(\mathcal{A})\right| \leq \mathsf{negl}(\lambda) \text{ and } \left|\mathsf{Adv}_1^{c2s}(\mathcal{A}) - \mathsf{Adv}_2^{c2s}(\mathcal{A})\right| \leq \mathsf{negl}(\lambda)$$

Thus, from now on if a client instance receives a valid first or third message, this message comes from a unique (see game 1) server instance. (We can indeed assert that it is the *exact same message* because we took the precaution of requiring that $(\mathsf{SKeygen}, \mathsf{Sig}, \mathsf{Ver})$ be *strongly*-EU-CMA-secure.)

Game 3. In this game, $\mathcal{C}^i$ computes its ciphertext as follows: Instead of encrypting $(K, N, M, \mathcal{C}, \mathcal{S}, pw_\mathcal{C})$, it encrypts $(1^\ell, N, M, \mathcal{C}, \mathcal{S}, 1^{\ell_{pw}})$.

When a server instance $\mathcal{S}^j$ receives a second protocol message m_2, $\mathcal{CH}$ examines m_2's origins. If m_2 was computed by some $\mathcal{C}^i$ that received and accepted $\mathcal{S}^j$'s first protocol message, $\mathcal{CH}$ simply accepts m_2 without decrypting and assigns $sid_\mathcal{S}^j \leftarrow M$ and $sk_\mathcal{S} \leftarrow K$, where M and K were chosen by $\mathcal{C}^i$. We know that this is well defined because the previous games make $\mathcal{C}^i$ unique if it exists. If m_2 was computed by some $\mathcal{C}^i$ that accepted some other protocol message, $\mathcal{CH}$ rejects without decrypting. Finally, if m_2 was adversary-generated, $\mathcal{CH}$ responds according to protocol specification.

The changes made to this game are computationally unnoticeable by $\mathcal{A}$ by virtue of the CCA-2 security of $(\mathsf{EKeyGen}, \mathsf{Enc}, \mathsf{Dec})$. Hence:

$$\left|\mathsf{Adv}_2^s(\mathcal{A}) - \mathsf{Adv}_3^s(\mathcal{A})\right| \leq \mathsf{negl}(\lambda) \text{ and } \left|\mathsf{Adv}_2^{c2s}(\mathcal{A}) - \mathsf{Adv}_3^{c2s}(\mathcal{A})\right| \leq \mathsf{negl}(\lambda)$$

(In order to make this "hop" fully rigorous, one must implement a decryption oracle to properly respond to the adversary-generated encryptions; this is the technical reason CCA-2 security is required.)

Note that from now on, the encryptions produced by client instances contain neither password information nor session key information.

Game 4. Let Bad be the event "a server instance accepts an adversary-generated message". Game 4 aborts and declares the adversary victorious if Bad occurs. Clearly:

$$\mathsf{Adv}_3^s(\mathcal{A}) \leq \mathsf{Adv}_4^s(\mathcal{A}) \text{ and } \mathsf{Adv}_3^{c2s}(\mathcal{A}) \leq \mathsf{Adv}_4^{c2s}(\mathcal{A})$$

Interlude: Where are we at with the SIDs? The modifications to the games played up until now show that if a client instance $\mathcal{C}^i$ accepts the third protocol message, then it must be partnered to some unique server instance. In particular, both instances are uniquely bound to the random SID $(M, \mathcal{S}, \mathcal{C})$, and hold the same SK $sk = K$. Similarly, if some server instance $\mathcal{S}^j$ accepts the second protocol message, it must be (at least) *semi*-partnered to some unique client instance $\mathcal{C}^i$. Again, the SIDs match and indeed bind the session to the instances. Thus, it really looks as though the SID is doing its job, which is to uniquely identify a session.

Finishing the proof. A server accepts an adversary generated message if and only if this message decrypts to $(pw, K, N, M, \mathcal{C}, \mathcal{S})$ for $\mathcal{C} = pid_\mathcal{S}^j$, some K and M in $\{0,1\}^\ell$, where N is $\mathcal{S}^j$'s nonce, and $pw = pw_\mathcal{C}$. In particular, we must have $pw = pw_\mathcal{C}$. But from the changes in game 3, there is no password information anywhere in $\mathcal{A}$'s view, so pw is independent of $pw_\mathcal{C}$. This implies that $\mathsf{Pr}[\mathsf{Bad}] \leq \frac{n_{se}}{|\mathsf{PW}|}$. Using this, we can evaluate $\mathsf{Adv}_4^s(\mathcal{A})$ and $\mathsf{Adv}_4^{c2s}(\mathcal{A})$.

We have $\mathsf{Pr}_4[\mathsf{S}] = \mathsf{Pr}_4[\mathsf{S}|\mathsf{Bad}]\mathsf{Pr}_4[\mathsf{Bad}] + \mathsf{Pr}_4[\mathsf{S}|\neg\mathsf{Bad}]\mathsf{Pr}_4[\neg\mathsf{Bad}]$. Since conditioned on Bad not having occurred all session keys are completely random and independent of $\mathcal{A}$'s view, we have $\mathsf{Pr}_4[\mathsf{S}|\neg\mathsf{Bad}] = \frac{1}{2}$. Combining this with the fact that the adversary wins if Bad occurs, we get $\mathsf{Pr}_4[\mathsf{S}] \leq \frac{1}{2} + \frac{1}{2}\mathsf{Pr}_4[\mathsf{Bad}] \leq \frac{1}{2} + \frac{n_{se}}{2|\mathsf{PW}|}$, which yields

$$\mathsf{Adv}_4^s \leq \frac{n_{se}}{|\mathsf{PW}|}$$

A similar argument using $\mathsf{C2S}$ gives us that

$$\mathsf{Adv}_4^{c2s} \leq \frac{n_{se}}{|\mathsf{PW}|}$$

Summing up all the terms from the previous games gives us the final result. ■

B BPR-style models revisited

In this appendix, we just gather all of our requirements for a complete PAKE model, including the uniqueness of partners. This is a formal recap of what has already been stated.

Principals and instances. An interactive game, indexed by the security parameter $\lambda \in \mathbb{N}$, is played between a challenger $\mathcal{CH}$ and an adversary $\mathcal{A}$. All of the algorithms considered are Probabilistic, Polynomial-Time (PPT) in λ.

At the beginning of the game, there is a fixed set of *principals* (or *users*), partitioned into non-empty sets of *clients* $\mathcal{C}$ and *servers* $\mathcal{S}$. Each client $\mathcal{C}$ is assigned a *password* $pw_{\mathcal{C}}$ drawn uniformly at random from some finite set PW of bitstrings. Each server $\mathcal{S}$ holds the full set of all clients' passwords $\{pw_{\mathcal{C}}\}_{\mathcal{C}}$.

Adversary $\mathcal{A}$ has oracle access - via the queries described below - to any number of *instances* $\mathcal{U}^i$ $(i \in \mathbb{N})$ of any principal $\mathcal{U}$. An instance of $\mathcal{U}^i$ represents an attempt $\mathcal{U}$ makes at running the PAKE protocol over the network fully controlled by $\mathcal{A}$. An instance's objective is to compute a *Session Key* (or SK) it believes it shares with an instance $\mathcal{V}^j$ of some other principal $\mathcal{V}$. This should happen only if $\mathcal{U}^i$ thinks it is partnered (see below) to $\mathcal{V}^j$.

At any point in time, an instance $\mathcal{U}^i$ may *terminate* (see below). By this time, $\mathcal{U}^i$ should have computed **1)** a *Partner Identity* (or PID) $pid^i_{\mathcal{U}}$, a **2)** *Session Identity* (SID) $sid^i_{\mathcal{U}}$, and **3)** a SK $sk^i_{\mathcal{U}}$. The PID is a bitstring indicating the identity of the instance with which $\mathcal{U}^i$ believes it has communicated with. The SID is a bitstring serving as an identifier for both the key exchange run that just occurred, and the session in which the computed SK will subsequently serve. Often the SID is always in practice set to being the ordered concatenation of all exchanged protocol messages, except possibly the last message. At any point in time an instance may also *abort* (see below), with no SK. Once an instance has *halted* (see below) it can no longer be reused.

Status of instances and partnering.

Halting. An instance halts if it stops sending and receiving messages, and ceases to compute anything.

Halting can either be "good" (i.e. with a SK) or "bad" (i.e. without a SK).

Accepting. An instance $\mathcal{U}^i$ accepts if and only if $sid^i_{\mathcal{U}}$ is set to a non-ε value. Accepting means that $\mathcal{U}^i$ believes it is holding enough information to compute a SK. The instance has not necessarily halted.

This formally includes the possibility that it may have actually computed the SK value, but is not yet willing ot use it.

Terminating. An instance $\mathcal{U}^i$ terminates if and only if $sk^i_{\mathcal{U}}$ is set to a non-ε value. If an instance terminates, it halts. Terminating means $\mathcal{U}$ believes it holds a good SK, and is now willing to use it in higher-level applications. $\mathcal{U}^i$ will no longer send nor receive PAKE protocol messages.

If an instance terminates, it accepts, or has previously accepted. In both cases, $sid^i_{\mathcal{U}}$ is set to a non-ε value and remains so. If an instance accepts, it has not necessarily terminated.

Aborting. An instance $\mathcal{U}^i$ aborts if it halts without having terminated.

In other words, it has stopped participating in the protocol exchange, and is unwilling to assign a value to $sk^i_{\mathcal{U}}$. Aborting can very well happen after accepting: Just imagine an instance holding a SK, but waiting for the last confirmation message to finally start using this SK.

Semi-partnering. $\mathcal{U}^i$ and $\mathcal{V}^j$ are semi-partnered if **1)** one is a client and one is a server, **2)** $pid^i_{\mathcal{U}} = \mathcal{V}$ and $pid^j_{\mathcal{V}} = \mathcal{U}$, **3)** $sid^i_{\mathcal{U}} \neq \varepsilon$, $sid^j_{\mathcal{V}} \neq \varepsilon$, and $sid^i_{\mathcal{U}} = sid^j_{\mathcal{V}}$.

Partnering. $\mathcal{U}^i$ and $\mathcal{V}^j$ are partnered if **1)** they are semi-partnered and **2)** $sk_{\mathcal{U}}^i \neq \varepsilon$, $sk_{\mathcal{V}}^j \neq \varepsilon$, and $sk_{\mathcal{U}}^i = sk_{\mathcal{V}}^j$.

By definition, if an instance is semi-partnered to another, it has accepted, and holds a SID. If it is partnered to another, it has terminated and holds a SK.

Partnering graph. A partnering graph is a graph with instances for nodes. Two nodes have an edge if and only if corresponding instances are partners.

Correctness. If $\mathcal{C}^i$ with $pid_{\mathcal{C}}^i = \mathcal{S}$ and $\mathcal{S}^j$ with $pid_{\mathcal{S}}^j = \mathcal{C}$ run the protocol fully and correctly, $\mathcal{C}^i$ and $\mathcal{S}^j$ are partnered.

Remark: Correctness can be formulated formally using *matching conversations* [6]. Note that correctness says that matching conversations lead to partnering. However, in full generality, just because two instances are partnered does not mean that they have had a matching conversation, although in practice for PAKEs this is often the case.

Find-then-Guess.

- $\mathsf{send}(\mathcal{U}, i, m)$: $\mathcal{A}$ has message m delivered to $\mathcal{U}^i$. $\mathcal{U}^i$ processes the message according to protocol specification. To instruct an instance $\mathcal{U}$ to send the first protocol message to entity $\mathcal{V}$, $\mathcal{A}$ makes the query with $M = \mathcal{V}$. This query is used to model arbitrary message delivery to an instance. In particular, it serves to count impersonation attacks.
- $\mathsf{execute}(\mathcal{U}, \mathcal{V}, i, j)$: The protocol is executed faithfully and completely between $\mathcal{U}^i$ and $\mathcal{V}^j$ and the resulting transcript is given to $\mathcal{A}$. $\mathcal{A}$ thus gets to see as many honest protocol runs as it wishes.
- $\mathsf{reveal}(\mathcal{U}, i)$: If $\mathcal{U}^i$ has not terminated or if it is part of a partnering graph in which an instance has been tested, the query returns $\perp$. Otherwise, it returns $sk_{\mathcal{U}}^i$ to $\mathcal{A}$.
- $\mathsf{test}(\mathcal{U}, i)$: $\mathsf{test}(\mathcal{U}, i)$: If $\mathcal{U}^i$ has not terminated or is not fresh, this returns $\perp$. Otherwise, $\mathcal{CH}$ flips a coin b outside of $\mathcal{A}$'s view. If $b = 0$, a random string R is drawn from the session key space and $tk_{\mathcal{U}}^i \leftarrow R$. Otherwise, $tk_{\mathcal{U}}^i \leftarrow sk_{\mathcal{U}}^i$. $tk_{\mathcal{U}}^i$ is then returned to $\mathcal{A}$. The test query may only be used once in the game.

Eventually, $\mathcal{A}$ halts the overall game, at which point it outputs a bit b'. If the game was halted without $\mathcal{A}$ making any test query, then $\mathcal{CH}$ privately flips a coin b.

Freshness: An instance $\mathcal{U}^i$ is said to be *fresh* if it is in a partnering graph in which no instance has been the target of a reveal query.

Real-or-Random. The send and $\mathsf{execute}$ queries are identical to those in the FtG model. However, the reveal query is no longer available. Instead, it is replaced by as many test queries as $\mathcal{A}$ wants. How they are answered depends on the value of a bit b flipped by $\mathcal{CH}$ outside of $\mathcal{A}$'s view *at the beginning of the game.*

- $\mathsf{test}(\mathcal{U}, i)$: If $\mathcal{U}^i$ has not terminated, $\perp$ is returned. Otherwise, suppose first that $b = 0$. If $\mathcal{U}^i$ is not partnered to any instance, or no instance in the partnering graph it is a part of was subjected to a test query, $\mathcal{CH}$ selects a random R from the session key space and sets $tk_{\mathcal{U}}^i \leftarrow R$. If $\mathcal{U}^i$ is within a partnering graph where some instance $\mathcal{V}^j$ that has been tested, $\mathcal{CH}$ sets $tk_{\mathcal{U}}^i \leftarrow tk_{\mathcal{V}}^j$. Suppose now that $b = 1$. In this case, $\mathcal{CH}$ sets $tk_{\mathcal{U}}^i \leftarrow sk_{\mathcal{U}}^i$. Then, $\mathcal{A}$ receives $tk_{\mathcal{U}}^i$.

The slight complication arising in case $b = 0$ is that even if the SKs assigned are random, they must at least remain consistent across instances that should hold the same keys. As in FtG, at any point in time $\mathcal{A}$ may halt the game and output a bit b'

Technically defining security. In both FtG and RoR, one usually considers three security properties: SK security, Client-to-Server (C2S) authentication, and Server-to-Client (S2C) authentication. We explicitly add to this uniqueness of partners by requiring that SIDs are shared by at most two instances (SID).

SK security: Let S be the event that "$b' = b$ at the end of the game". Event S measures the *semantic security of the SK*, i.e. the adversary's ability to tell this key apart from a random string. Since $\mathcal{A}$ can simply flip a coin to get the answer right with probability $1/2$, $\mathcal{A}$'s natural advantage is defined to be $\mathsf{Adv}^s(\mathcal{A}) := 2\mathsf{Pr}[\mathsf{S}] - 1$. A PAKE protocol is said to have semantically secure SKs if there exists a non-zero constant $C \in \mathbb{N}$ such that for any PPT $\mathcal{A}$, there exists a negligible (in λ) function negl with the property that $\mathsf{Adv}^s(\mathcal{A}) \leq \frac{Cn_{se}}{|\mathsf{PW}|} + \mathsf{negl}(\lambda)$ where n_{se} is an upper bound on the number of send queries the adversary makes.

Authentication: Let $\mathsf{C2S}$ be the event that "there exists some server instance $\mathcal{S}^j$ that is ready to use a SK, but has not had a correct exchange with a client instance." This event measure the adversary's ability to cause client-to-server authentication to fail, i.e. a server thinks it is talking to a correct client when it is not. Here we simply set $\mathsf{Adv}^{c2s}(\mathcal{A}) := \mathsf{Pr}[\mathsf{C2S}]$, and we say that a PAKE achieves client-to-server authentication if there exists a non-zero C such that for any PPT $\mathcal{A}$, there exists a negligible function negl with the property that $\mathsf{Adv}^{c2s}(\mathcal{A}) \leq \frac{Cn_{se}}{|\mathsf{PW}|} + \mathsf{negl}(\lambda)$. Server-to-client authentication is defined similarly.

Partner uniqueness: Let SID be the event that "there exists more than two instances that have the same non-ε SID". Let $\mathsf{Adv}^{sid}(\mathcal{A}) := \mathsf{Pr}[\mathsf{SID}]$. We say that PAKE achieves unique partnering if for any PPT $\mathcal{A}$ the function Adv^{sid} is negligible. Note that the quality of the long-term keying material here no longer should have an influence on the security bound we require.

Of the four afore-mentioned security properties, our discussion implies that the first and last are always required.