A Universally Composable PAKE with Zero Communication Cost

(And Why It Shouldn't Be Considered UC-Secure)

Lawrence Roy[1] and Jiayu Xu[2]

[1] Aarhus University, ldr709@gmail.com
[2] Oregon State University, xujiay@oregonstate.edu

Abstract. A Password-Authenticated Key Exchange (PAKE) protocol allows two parties to agree upon a cryptographic key, when the only information shared in advance is a low-entropy password. The standard security notion for PAKE (Canetti et al., Eurocrypt 2005) is in the Universally Composable (UC) framework. We show that unlike most UC security notions, UC PAKE does not imply correctness. While Canetti et al. has briefly noticed this issue, we present the first comprehensive study of correctness in UC PAKE:

1. We show that TrivialPAKE, a no-message protocol that does not satisfy correctness, is a UC PAKE;
2. We propose nine approaches to guaranteeing correctness in the UC security notion of PAKE, and show that seven of them are equivalent, whereas the other two are unachievable;
3. We prove that a direct solution, namely changing the UC PAKE functionality to incorporate correctness, is impossible;
4. Finally, we show how to naturally incorporate correctness by changing the model — we view PAKE as a three-party protocol, with the man-in-the-middle adversary as the third party.

In this way, we hope to shed some light on the very nature of UC-security in the man-in-the-middle setting.

1 Introduction

A *password-authenticated key exchange (PAKE)* protocol allows two parties to jointly establish a cryptographically strong key, where the only information shared in advance is a low-entropy password. Crucially, such protocols must remain secure in the presence of a man-in-the-middle adversary. Since its first proposal in 1992 [5], PAKE protocols have been extensively studied in various security models and under various assumptions; a very incomplete list of works includes [1, 3, 4, 7, 10, 18, 22, 23]. Interest in the deployment of PAKE protocols in practice — especially their integration with TLS — has been on the rise in recent years, culminating in the standardization process by the IRTF in 2019–20 [13, 26]. Several classes of extensions, such as asymmetric PAKE [6, 20] and fuzzy PAKE [14], have also been considered.

Security definitions for PAKE. Since passwords have low entropy, any security definition of PAKE has to take into account the fact that an adversary has non-negligible probability of guessing it correctly. Roughly speaking, the basic security property of PAKE is that the only feasible attack is via online guessing, whose probability of success is $1/|\mathsf{Dict}|$ per session (where Dict is the password dictionary, i.e., the set of all possible passwords). In particular, offline dictionary attacks, where the adversary performs a brute-force search over the dictionary upon seeing protocol messages, must be prevented.

There are two major paradigms of PAKE security definitions: game-based [4] and Universally Composable (UC) [10]. In multi-party computation, the UC definition is generally preferable since it supports arbitrary composition, namely the security of PAKE is preserved when composed with itself or other protocols, sequentially or in parallel. In the context of PAKE, the UC definition has the additional advantage that it naturally takes reused password or correlated passwords into consideration, which is difficult to model in the game-based setting since the latter is stand-alone in nature. The UC definition has become the de facto standard of PAKE security; in particular, all candidates in the second round of the IRTF standardization competition have a UC security analysis [1, 2, 18, 20].

UC PAKE and correctness. A cryptographic protocol usually needs to satisfy some notion of correctness (sometimes called completeness), namely the parties' outputs meet some desired requirements when there is no attack. For a PAKE protocol, correctness means that the two parties should output the same key as long as their passwords are equal. In the game-based definitions, correctness and security are usually defined separately. By contrast, in the UC setting, correctness is often a trivial implication of UC-security; that is, if we remove the ideal adversary and let all protocol parties be honest in the UC functionality and observe their outputs, then correctness can be seen immediately from the functionality's code. This the case for e.g., universally composable commitment schemes [9] and oblivious transfer protocols [12].

Somewhat surprisingly, we show that for PAKE protocols, UC-security does not imply correctness. In particular, in Section 3 we show protocol TrivialPAKE, where the two parties independently output random keys, is a UC PAKE. At a high level, this is because the UC PAKE functionality allows the ideal adversary to cause the two protocol parties to output independent keys, even if both parties are honest; therefore, a simulator can leverage such mechanism to complete the simulation for TrivialPAKE. Of course, the same argument also goes for protocols in which the two parties communicate in some arbitrary manner, and then output independent random keys.

We note that the original UC PAKE paper [10] has already noticed that UC PAKE does not imply correctness (called non-triviality therein). In [10, Section 7], the authors wrote:

> *A protocol is non-trivial if two honest parties are ensured to agree on matching session keys at the conclusion of a protocol execution (except perhaps with negligible probability), provided that (1) both parties use*

the same password, and (2) the adversary passes all messages between the parties without modifying them or inserting any messages of its own. The non-triviality requirement is needed since the "empty" protocol where parties do nothing securely realizes $\mathcal{F}_{\mathsf{pwKE}}$ (the ideal-model simulator simply never issues a NewKey *query to the functionality and so the parties never actually obtain keys).*

However, the "empty" protocol realizes *any* two-party UC functionality, and can be easily ruled out by requiring both parties to output *something*. (Indeed, this is the approach suggested by [11, Section 2] in the context of generic two-party computation.) By contrast, the issue with our TrivialPAKE is unique to PAKE. We will see in Section 4.1 that the underlying mechanisms of these two counterexamples are different: one is because the simulator might send some commands while it should not (in TrivialPAKE), the other is because the simulator might not send some commands while it should (in the "empty" protocol).

Furthermore, the fact that correctness needs to be checked separately from UC-security appears under-appreciated: some works on UC PAKE do not perform this correctness check [15, 17, 20, 25], which is sometimes not completely trivial.[1]

Guaranteeing correctness. At first glance, it seems trivial to fix the definition of a UC PAKE: just require that the protocol satisfy correctness as well as realize the PAKE ideal functionality. However, we believe a deeper understanding of this issue is warranted, to learn *why* the PAKE functionality fails to guarantee correctness (as opposed to most UC functionalities that do imply correctness) and *how* to best address the issue. In Section 4, we study several approaches to enforcing the correctness requirement in UC PAKE, finding that some are impossible to achieve, while the rest are equivalent.

These definitions come from two different styles. First, as mentioned above, we could enforce correctness separately from UC-security; namely, we require that two honest parties using the same password, without any adversary interference, must get the same key at the end. However, while UC definitions are based on a correspondence between a real world and an ideal world, the aforementioned definition of correctness only involves the real world; as a result, it fails to provide any kind of composability, which makes it harder to

[1] The cases of [17, 20, 25] are especially problematic, since their (asymmetric) PAKE protocols use a UC-secure authenticated key exchange (AKE) protocol as a building block, and their modelings of UC AKE also do not guarantee correctness. Therefore, a proof of PAKE correctness would need a separate correctness notion for AKE: roughly, that there exists an efficient algorithm Gen such that if we run it twice and obtain two key pairs (pk, sk) and (pk', sk'), then two AKE parties with inputs (pk, pk', sk) and (pk', pk, sk') should output the same key (assuming there is no adversarial interference). Such structural requirement is not mentioned in any of the aforementioned works (and in particular, the Key.Gen algorithm in [25, Fig. 6] is undefined). The exact requirement of AKE correctness used in [17, 20, 25] is out of the scope of this work.

either prove that a protocol is a valid PAKE (such as for [17, 20, 25][1]), or to make use of this proof in a higher level protocol — that is, the correctness of the higher level protocol always needs to be proved separately.

Next, we consider placing constraints on the UC simulator for PAKE. The basic observation here is that the simulator for TrivialPAKE always interrupts the protocol sessions, causing the two protocol parties to output independent random keys, even though such "interruption" mechanism in the PAKE functionality is meant only to model a man-in-the-middle adversary that modifies protocol messages. In other words, the underlying reason why TrivialPAKE is UC-secure is that the simulator is given too much power. We consider two slightly different ways to disallow such "rogue" interruptions of protocol sessions: requiring either a so-called *reasonable simulator* or a *strong reasonable simulator*. That is, we show that the existence of a (strong) reasonable simulator is equivalent to the PAKE protocol being correct. We further prove that a seemingly stronger notion, namely *all* successful simulators must be (strong) reasonable simulators, is also equivalent to the PAKE protocol being correct. Finally, we consider the question of whether the simulator can perform a "rogue" interruption with negligible or zero probability; we call the latter kind a *(strong) perfectly reasonable simulator*, and show that PAKE correctness is equivalent to the existence of a (strong) perfectly reasonable simulator. However, for any correct PAKE there always exist simulators that fail to be perfectly reasonable. In sum, we present eight approaches to placing requirements on the simulator, and show that six of them are equivalent to directly enforcing PAKE correctness, whereas the other two are unachievable. Finally, we note that this definition style has been used to address a separate issue in the context of *asymmetric* PAKE [15, 20]; however, the necessity of similar constraints on simulators for (regular) PAKE protocols went unnoticed.

Placing constraints on the simulator has the advantage that they often compose easily (a similar observation is made in [15] in the context of asymmetric PAKE). Still, such solutions are ad-hoc and we might ask whether we can just modify the PAKE ideal functionality so that it implies correctness. Unfortunately, in Section 5 we show that this is impossible. The problem is that the ideal functionality has no clue whether the adversary is tampering with the PAKE messages; therefore, it must allow the simulator to interrupt the session in case the adversary interferes with the PAKE protocol in the real world, but then the simulator can always use this interface to stop the keys from matching.

Finally, in Section 6 we show how a more sophisticated model bypasses the impossibility result and allows correctness to be included in the UC PAKE functionality, making it fully composable. We add a third party called the *router*, which is connected to both protocol parties via an authenticated channel; when the router is honest, it simply forwards messages between the two protocol parties. (Of course, a corrupted router is free to deviate from its description arbitrarily, namely it can modify the messages sent between the

two protocol parties.) Correspondingly, in our modified PAKE ideal functionality we require that session interruption must be done by a *corrupted party*, who must be one of the three participants (including the router), rather than the UC simulator. We prove that our notion of UC three-party PAKE is equivalent to the definition of UC PAKE plus correctness.

2 Preliminaries

Notations. For any $q \in \mathbb{N}^+$, define $[q]$ as the set $\{1, \ldots, q\}$. We let λ denote the security parameter. For a set X, let $x \leftarrow X$ denote the process of sampling an element x uniformly at random from X. We use "efficient" as a shorthand for "probabilistic polynomial time".

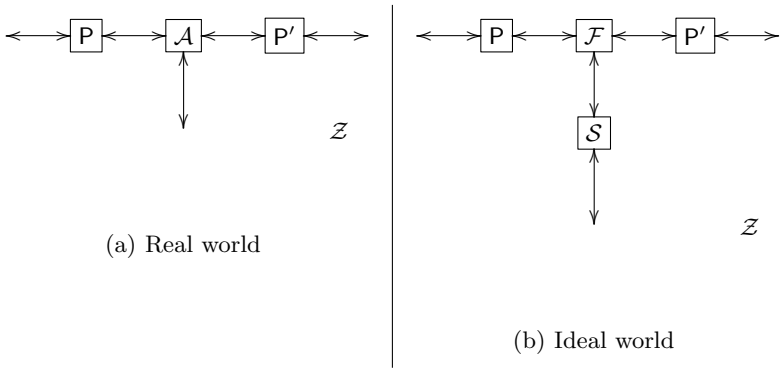
2.1 Overview of the UC Framework

In this section, we briefly review the Universally Composable framework by Canetti [8], and introduce necessary notations to be used in later sections. For simplicity's sake, we only cover two-party protocols, and it extends to the case of multi-party protocols (used in Section 6 only) naturally.

A *protocol* Π involves two parties (modeled as efficient interactive Turing machines), P and P', sending messages to each other. In an *execution* of protocol Π, there are two additional parties, the *environment* $\mathcal{Z}$ and the *adversary* $\mathcal{A}$, where $\mathcal{Z}$ sends inputs to P and P' and receives outputs from them; furthermore, $\mathcal{Z}$ and $\mathcal{A}$ may communicate with each other at any time during protocol execution. Multiple sessions may run in parallel during protocol execution, distinguished by a *session id* denoted *sid* (which is agreed upon before protocol execution and is not part of the protocol description). For a PAKE protocol, we consider the man-in-the-middle setting, where all messages sent between P and P' pass the adversary $\mathcal{A}$, which can arbitrarily modify these messages or simply drop them. In the special case where $\mathcal{A}$ merely transmits all messages between P and P' without modifying or dropping any of them, we say $\mathcal{A}$ is an *eavesdropper*.

The above describes the *real world*. In the *ideal world*, there is an uncorruptable *ideal functionality* $\mathcal{F}$ whose code is public, and the adversary $\mathcal{A}$'s role is replaced by an *ideal adversary* (a.k.a. *simulator*) $\mathcal{S}$. $\mathcal{F}$ communicates with protocol parties P and P', as well as the ideal adversary $\mathcal{S}$; P and P' are "dummy" parties that merely transmit messages between $\mathcal{F}$ and $\mathcal{Z}$ without any modifications. Importantly, $\mathcal{F}$ and $\mathcal{Z}$ do not communicate with each other directly.

In both the real world and the ideal world, the view of the environment $\mathcal{Z}$ consists of its input to/output from protocol parties P and P', as well as its communications with either the (real) adversary $\mathcal{A}$ or the ideal adversary $\mathcal{S}$. In a nutshell, UC-security says that any efficient environment's view can be successfully simulated by an efficient simulator, meaning that $\mathcal{Z}$ cannot distinguish whether it is in the real world or the ideal world:



(a) Real world

(b) Ideal world

Fig. 1: Real world and ideal world in the UC framework

Definition 1. *For a protocol* Π*, an ideal functionality* $\mathcal{F}$*, and a (real) adversary* $\mathcal{A}$*, we say a simulator* $\mathcal{S}$ *is* successful *w.r.t.* $\mathcal{A}$ *if (1)* $\mathcal{S}$ *is efficient, and (2) for any efficient environment* $\mathcal{Z}$*,*

$$\begin{aligned}\mathbf{Dist}_{\Pi,\mathcal{F}}(\mathcal{A},\mathcal{S},\mathcal{Z}) \triangleq |\Pr[&\mathcal{Z} \text{ outputs } 1 \text{ in the real world with } \Pi,\mathcal{A}]\\ &- \Pr[\mathcal{Z} \text{ outputs } 1 \text{ in the ideal world with } \mathcal{F},\mathcal{S}]|\end{aligned}$$

is negligible, where the probability is taken over the randomness generated in the execution of Π*, as well as the random tapes of* $\mathcal{F}$*,* $\mathcal{A}$*,* $\mathcal{S}$*, and* $\mathcal{Z}$*.*

Definition 2. *We say protocol* Π UC-realizes *functionality* $\mathcal{F}$ *if for any efficient adversary* $\mathcal{A}$*, there exists a successful simulator w.r.t.* $\mathcal{A}$*.*

A standard result [8, Claim 11] states that only the dummy adversary, i.e., the adversary that merely transmits all messages between the environment and the protocol parties, needs to be considered. (We may intuitively say that $\mathcal{A}$ follows $\mathcal{Z}$'s "instructions", e.g., $\mathcal{Z}$ "instructs" $\mathcal{A}$ to send message (sid, m) to protocol party P.) That is, Definition 2 is equivalent to the following:

Definition 3. *We say protocol* Π UC-realizes *functionality* $\mathcal{F}$ *if there exists a successful simulator w.r.t. the dummy adversary.*

Definition 3 is what we will use in subsequent sections. Since the adversary $\mathcal{A}$ is now fixed, we may simply say "the simulator $\mathcal{S}$ is successful" and write $\mathbf{Dist}_{\Pi,\mathcal{F}}(\mathcal{S},\mathcal{Z})$. Furthermore, when Π and $\mathcal{F}$ are clear from context, we may drop them and write $\mathbf{Dist}(\mathcal{S},\mathcal{Z})$.

2.2 Overview of PAKE

A *password-authenticated key exchange (PAKE)* is a two-party protocol where each party inputs a supposedly low-entropy string (called the *password*) and outputs a cryptographic *session key*. Let Dict be the set of all candidate

passwords; we place no restrictions on $|\mathsf{Dict}|$ except that $|\mathsf{Dict}| \geq 2$. Correctness requires that if the two parties' passwords match, and there is no man-in-the-middle attack, then they output a shared key with overwhelming probability:

Definition 4. *We say a PAKE protocol* Π *is* correct *if the following holds: for any* $\mathsf{pw} \in \mathsf{Dict}$*, in an execution of* Π*, if both protocol parties* P *and* P' *input* (sid, pw) *(with appropriate additional fields for UC-compatibility; see Figure 2 below), both* P *and* P' *are honest, and the adversary* $\mathcal{A}$ *is an eavesdropper, then*

$$\Pr[\mathsf{Correct}(\mathsf{pw})] \triangleq \Pr[\mathsf{P} \text{ outputs } (sid, K) \wedge \mathsf{P}' \text{ outputs } (sid, K') \wedge K = K' \in \{0,1\}^\lambda]$$

is overwhelming, where the probability is taken over the randomness generated in the execution of Π*.*

Note that our notion of correctness requires that P and P' must output *something* at the end of the session. This is a trivial requirement since the parties can always output a random key (or an "abort" symbol in the case of explicit authentication; see discussion at the end of this section).

The UC PAKE functionality. We recall in Figure 2 the standard UC PAKE functionality $\mathcal{F}_{\text{PAKE}}$ from [10] (with minor notational changes).

- On input $(\mathsf{NewSession}, sid, \mathsf{P}, \mathsf{P}', \mathsf{pw}, \mathsf{role})$ from P, send $(\mathsf{NewSession}, sid, \mathsf{P}, \mathsf{P}', \mathsf{role})$ to $\mathcal{S}$. Furthermore, if this is the first NewSession message for sid, or this is the second NewSession message for sid and there is a record $\langle \mathsf{P}', \mathsf{P}, \cdot \rangle$, then record $\langle \mathsf{P}, \mathsf{P}', \mathsf{pw} \rangle$ and mark it fresh.
- On $(\mathsf{TestPwd}, sid, \mathsf{P}, \mathsf{pw}^*)$ from $\mathcal{S}$, if there is a record $\langle \mathsf{P}, \mathsf{P}', \mathsf{pw} \rangle$ marked fresh, then do:
  - If $\mathsf{pw}^* = \mathsf{pw}$, then mark the record compromised and send "correct guess" to $\mathcal{S}$.
  - If $\mathsf{pw}^* \neq \mathsf{pw}$, then mark the record interrupted and send "wrong guess" to $\mathcal{S}$.
- On $(\mathsf{NewKey}, sid, \mathsf{P}, K^* \in \{0,1\}^\lambda)$ from $\mathcal{S}$, if there is a record $\langle \mathsf{P}, \mathsf{P}', \mathsf{pw} \rangle$, and this is the first NewKey message for sid and P, then output (sid, K) to P, where K is defined as follows:
  - If the record is compromised, or either P or P' is corrupted, then set $K := K^*$.
  - If the record is fresh, a key (sid, K') has been output to P', at which time there was a record $\langle \mathsf{P}', \mathsf{P}, \mathsf{pw} \rangle$ marked fresh, then set $K := K'$.
  - Otherwise sample $K \leftarrow \{0,1\}^\lambda$.

  Finally, mark the record completed.

Fig. 2: UC PAKE functionality $\mathcal{F}_{\text{PAKE}}$

The functionality allows for three types of commands:

– A NewSession command, sent from a protocol party P, indicates that P (whose password is pw) wants to jointly establish a key with another party P′.[2] This initiates a session from P to P′; for each session id *sid*, only one session from P to P′ and one session from P′ to P is allowed. The NewSession command is transmitted to the ideal adversary $\mathcal{S}$ (without the password pw), which corresponds to the real-world scenario where the adversary sees the first message from P to P′ and thus learns that the P → P′ session has started.
– A TestPwd command models the inevitable attack in which the adversary chooses a password guess pw* and communicates with P by running the algorithm of P′ on pw*. If pw* happens to be the password of P (i.e., the adversary's password guess is correct), then the adversary learns the session key of P when the session ends; otherwise the adversary should not learn anything about the session key of P.[3] Thus, $\mathcal{F}_{\text{PAKE}}$ marks the P → P′ session record compromised (for a correct guess) or interrupted (for a wrong guess) accordingly. Importantly, once the session record becomes compromised or interrupted, all further TestPwd commands for the same session will be ignored; this implies that the adversary can only test one password for the P → P′ session and one password for the P′ → P session.
– Finally, a NewKey command models the end of a session where the party outputs a session key. How the session key is determined depends on the status of the session:
  - If *both* the P → P′ session and the P′ → P session are fresh, i.e., the adversary did not interrupt the communication between P and P′, then the two parties should output the same key as long as their passwords match; furthermore, the key should be random to the adversary. This is formally modeled as follows: assume w.l.o.g. that P receives its session key before P′ does. Then P should output a random key (modeled in the third case under NewKey), and when P′ outputs its session key, the key should be equal to what was previously output by P (modeled in the second case under NewKey).
  - If the session is compromised, i.e., the adversary successfully guessed the password during an online attack, then as noted above, the adversary learns the session key. In this case, we consider all security guarantees to

[2] The role field might be necessary for the description of the protocol when the algorithms of the two parties are not completely identical (especially when one party must wait for the other party's message in order to start its own session; see [10, Figure 5] for an example), but it has nothing to do with the security of the protocol.

[3] The functionality in Figure 2 lets the ideal adversary $\mathcal{S}$ learn whether its password guess is correct or not. This is necessary for the simulation of some PAKE protocols but not for others. The variant where $\mathcal{S}$ does not learn this information is called *implicitly-only PAKE*; see [14] for further discussion. We follow the standard PAKE functionality, but note that all theorems below apply to implicitly-only PAKE as well.

be lost, so we might as well let the ideal adversary $\mathcal{S}$ choose the session key. This is modeled in the first case under NewKey.

- If the session is interrupted, i.e., the adversary performed an online attack using a wrong password guess, then as noted above, the session key should be independent of the adversary's view. The same goes for the case where the session $\mathsf{P} \to \mathsf{P}'$ is fresh but its counter-session $\mathsf{P}' \to \mathsf{P}$ has been attacked (either compromised or interrupted); as well as the case where both the $\mathsf{P} \to \mathsf{P}'$ session and the $\mathsf{P}' \to \mathsf{P}$ session are fresh, yet the passwords of P and P' are different. This is modeled in the third case under NewKey.

After outputting the session key, the session record is marked completed to prevent NewKey from being sent twice and TestPwd from being sent after the session ends.

The functionality above only achieves *implicit authentication*, namely if a session is interrupted by the adversary (or the passwords do not match), then the two parties do not learn this fact and merely output independent random keys. In a PAKE with *explicit authentication*, the two parties output an "abort" symbol instead. Explicit authentication can be achieved by adding one round to an implicit-authentication PAKE [16].

3 A No-Message UC PAKE

In this section, we consider protocol TrivialPAKE, where each party simply outputs a random string as the key. For a formal presentation in the UC framework, see Figure 3.

1. On input $(\mathsf{NewSession}, sid, \mathsf{P}, \mathsf{P}', \mathsf{pw}, \mathsf{role})$, if this is the first NewSession message for sid, party P samples $K \leftarrow \{0,1\}^\lambda$ and outputs (sid, K).

Fig. 3: Protocol TrivialPAKE

Obviously TrivialPAKE does not satisfy correctness. On the other hand, we show:

Proposition 1. *Protocol TrivialPAKE (Figure 3) realizes $\mathcal{F}_{PAKE}$.*

Proof. For the dummy adversary $\mathcal{A}$, construct simulator $\mathcal{S}$ as follows:

Simulator $\mathcal{S}$:

1. On $(\mathsf{NewSession}, sid, \mathsf{P}, \mathsf{P}', \mathsf{role})$ from $\mathcal{F}_{\text{PAKE}}$, send $(\mathsf{TestPwd}, sid, \mathsf{P}, \perp)$ to $\mathcal{F}_{\text{PAKE}}$ followed by $(\mathsf{NewKey}, sid, \mathsf{P}, 0^\lambda)$.

We claim that $\mathcal{S}$'s simulation is perfect, i.e., any environment $\mathcal{Z}$'s views in the real world and the ideal world are identical. Suppose $\mathcal{Z}$ inputs $(\mathsf{NewSession}, sid, \mathsf{P}, \mathsf{P}', \mathsf{pw}, \mathsf{role})$ to party P.[4] In the ideal world, $\mathcal{F}_{\mathrm{PAKE}}$ stores a record $\langle \mathsf{P}, \mathsf{P}', \mathsf{pw} \rangle$ and marks it fresh. When $\mathcal{F}_{\mathrm{PAKE}}$ receives $(\mathsf{TestPwd}, sid, \mathsf{P}, \bot)$, since $\mathsf{pw} \neq \bot$, the status of the record is changed to $\mathsf{interrupted}$. Finally, when $\mathcal{F}_{\mathrm{PAKE}}$ receives $(\mathsf{NewKey}, sid, \mathsf{P}, 0^\lambda)$, it enters the third case, so P receives a random string $K \leftarrow \{0,1\}^\lambda$ (independent of everything else) and outputs (sid, K). We conclude that in the ideal world, each party independently outputs a random key in $\{0,1\}^\lambda$ together with the session id — which is exactly the case in the real world. (Note that there are no protocol messages, so the only strings $\mathcal{Z}$ receives are parties' outputs.) This completes the proof. ■

Remark 1. One might hope to prevent TrivialPAKE from being a UC PAKE by simply disallowing the simulator from sending $(\mathsf{TestPwd}, sid, \cdot, \bot)$ to $\mathcal{F}_{\mathrm{PAKE}}$, i.e., $\mathcal{F}_{\mathrm{PAKE}}$ would ignore a $(\mathsf{TestPwd}, sid, \cdot, x)$ message if $x \notin \mathsf{Dict}$. Assuming $|\mathsf{Dict}|$ is polynomial and the password is chosen uniformly at random from Dict, the simulator always has non-negligible probability of guessing the password correctly (hence setting the session record $\mathsf{compromised}$). However, a simulator that sends $(\mathsf{TestPwd}, sid, \mathsf{P}, x)$ for any $x \in \mathsf{Dict}$ followed by $(\mathsf{NewKey}, sid, \mathsf{P}, K)$ for $K \leftarrow \{0,1\}^\lambda$, still ensures that each party independently outputs a random key (if $x = \mathsf{pw}$, P outputs K; if $x \neq \mathsf{pw}$, P outputs a random key freshly sampled by $\mathcal{F}_{\mathrm{PAKE}}$), thus its simulation is perfect.

4 Seven Equivalent Ways to Guarantee Correctness

4.1 Three Equivalent Ways to Guarantee Correctness

While TrivialPAKE is indeed trivial and the issue appears minor, closer scrutiny shows that there are a number of natural ways to guarantee correctness, resulting from different insights on the essence of the issue. In this section, we propose three such approaches, and show that they are equivalent.

First of all, a straightforward approach would be explicitly requiring correctness:

Proposal 1. Only consider PAKE protocols that both realize $\mathcal{F}_{\mathrm{PAKE}}$ and are correct.

It is instructive, however, to understand why $\mathcal{F}_{\mathrm{PAKE}}$ fails to guarantee correctness. Correctness is meant to be enforced in the second case under NewKey, where two parties output the same key if their passwords match and both of their session records are fresh — the latter of which in turn models an eavesdropping adversary. However, as we have seen in the proof of

[4] We can assume w.l.o.g. that $\mathcal{Z}$ never reuses a session id, i.e., it never sends $(\mathsf{NewSession}, sid, \mathsf{P}, \cdot, \cdot, \cdot)$ to P twice (otherwise the second message will be ignored in both the real world and the ideal world).

Proposition 1, there is a gap in the modeling, namely *the ideal adversary can interrupt the protocol sessions (causing the two parties to output independent keys) even if the real adversary merely eavedrops.* In other words, the ideal adversary is too strong, resulting in an unreasonably weak functionality.

To bridge this gap, we could limit the ideal adversary's power (thus making the power of the ideal adversary and the real adversary "equal") by enforcing the following rule: the ideal adversary is forbidden from sending $\mathsf{TestPwd}$, if the real adversary merely eavesdrops. To formalize this, we borrow some notations from [10]. Consider any PAKE protocol Π, and fix a simulator $\mathcal{S}$ for the dummy adversary $\mathcal{A}$. For any environment $\mathcal{Z}$, define $\mathsf{SpuriousGuess}(\mathcal{S}, \mathcal{Z})$ as the following event: both P and P' are honest, there exists a session *sid* in which $\mathcal{A}$ is an eavesdropper, yet $\mathcal{S}$ sends a $(\mathsf{TestPwd}, sid, \cdot, \cdot)$ message to $\mathcal{F}_{\text{PAKE}}$. As a separate condition, define $\mathsf{NoOutput}(\mathcal{S}, \mathcal{Z})$ as the following event: both P and P' are honest, there exists a session *sid* in which $\mathcal{S}$ receives $(\mathsf{NewSession}, sid, \mathsf{P}, \cdot, \cdot)$ from $\mathcal{F}_{\text{PAKE}}$, yet $\mathcal{S}$ does not send $(\mathsf{NewKey}, sid, \mathsf{P}, K \in \{0,1\}^\lambda)$ to $\mathcal{F}_{\text{PAKE}}$ before it halts.

Definition 5. *We say a simulator $\mathcal{S}$ is* reasonable *if for any efficient environment $\mathcal{Z}$,* $\Pr[\mathsf{SpuriousGuess}(\mathcal{S}, \mathcal{Z})]$ *and* $\Pr[\mathsf{NoOutput}(\mathcal{S}, \mathcal{Z})]$ *are both negligible, where the probability is taken over the randomness generated in the execution of Π, as well as the random tapes of $\mathcal{S}$, $\mathcal{Z}$, and $\mathcal{F}_{PAKE}$.*[5]

We can now consider a proposal where only reasonable simulators "count", and unreasonable simulators (such as the one in the proof of Proposition 1) are considered invalid:

Proposal 2. Only consider PAKE protocols for which a successful reasonable simulator exists.

The above proposal does not rule out the possibility that for some protocols, there are some reasonable simulators and some unreasonable simulators. We could strengthen it to:

Proposal 3. Only consider PAKE protocols for which **all** successful simulators are reasonable.

Which, then, is the "right" proposal to guarantee correctness? We now show that all three are the "right" approach, since they are equivalent:

Lemma 1. *Let Π be any PAKE protocol. Then the followings are equivalent:*

(1) Π is correct and realizes $\mathcal{F}_{PAKE}$;
(2) There exists a successful reasonable simulator for Π;

[5] The purpose of considering $\mathsf{NoOutput}$ is to rule out the "empty" protocol mentioned in Section 1, where P and P' simply don't do anything (and $\mathcal{S}$ also doesn't do anything). Although our primary focus is to rule out protocols like TrivialPAKE, with the requirement on $\mathsf{NoOutput}$ in place and with how we define correctness (Definition 4), we can formally rule out the "empty" protocol as well.

(3) Π *realizes* $\mathcal{F}_{PAKE}$*, and all successful simulators for* Π *are reasonable.*[6]

Proof. (1)⇒(3): This is [10, Lemma A.1], except that [10] requires correctness to be perfect. For completeness, we present the full proof in Appendix A.

(3)⇒(2): This is immediate.

(2)⇒(1): The intuition is that, a reasonable simulator cannot send $\mathsf{TestPwd}$ in a session where the adversary merely eavesdrops, so in the ideal world this session must remain fresh, hence $\mathcal{F}_{\text{PAKE}}$ will let the two parties output the same key. This must happen in the real world as well (since the simulator is successful), which implies correctness.

Formally, let $\mathcal{S}$ be a successful reasonable simulator as in the statement of (2). Since $\mathcal{S}$ is successful, it follows that Π realizes $\mathcal{F}_{\text{PAKE}}$. We now show that Π is correct.

For any $\mathsf{pw} \in \mathsf{Dict}$, consider the following environment $\mathcal{Z}$:

Environment $\mathcal{Z}$:

1. Initialize a single session between P and P' on pw. That is, pick any sid, and input $(\mathsf{NewSession}, sid, \mathsf{P}, \mathsf{P}', \mathsf{pw}, \mathsf{role})$ to P and $(\mathsf{NewSession}, sid, \mathsf{P}', \mathsf{P}, \mathsf{pw}, \mathsf{role}')$ to P'.
2. Instruct $\mathcal{A}$ to be an eavesdropper in session sid.
3. When P outputs (sid, K) and P' outputs (sid, K'), output 1 if $K = K' \in \{0,1\}^\lambda$ and output 0 otherwise. If P or P' does not output anything when it halts, then output 0.

Since $\mathcal{S}$ is reasonable, we know that

$$\Pr[\mathsf{SpuriousGuess}(\mathcal{S}, \mathcal{Z})] = \Pr[\mathcal{S} \text{ sends } (\mathsf{TestPwd}, sid, \cdot, \cdot) \text{ to } \mathcal{F}_{\text{PAKE}}]$$

is negligible (where the equation is due to the fact that there is only one session, and $\mathcal{A}$ is an eavesdropper in this session). Suppose $\mathsf{SpuriousGuess}(\mathcal{S}, \mathcal{Z})$ does not occur. Then in the ideal world, when $\mathcal{S}$ sends $(\mathsf{NewKey}, sid, \mathsf{P}, \cdot)$ and $(\mathsf{NewKey}, sid, \mathsf{P}', \cdot)$ to $\mathcal{F}_{\text{PAKE}}$ (note that $\mathcal{S}$ must send such commands except with negligible probability $\Pr[\mathsf{NoOutput}(\mathcal{S}, \mathcal{Z})]$), $\mathcal{F}_{\text{PAKE}}$'s records $\langle \mathsf{P}, \mathsf{P}', \mathsf{pw} \rangle$ and $\langle \mathsf{P}', \mathsf{P}, \mathsf{pw} \rangle$ are both fresh, resulting in P and P' outputting the same key $K = K'$ (together with sid), as can be seen from the second case of $\mathcal{F}_{\text{PAKE}}$.[7] This causes $\mathcal{Z}$ to output 1. Therefore,

$$\Pr[\mathcal{Z} \text{ outputs 1 in the ideal world}] \geq 1 - \Pr[\mathsf{SpuriousGuess}(\mathcal{S}, \mathcal{Z})] - \Pr[\mathsf{NoOutput}(\mathcal{S}, \mathcal{Z})].$$

[6] The condition "Π realizes $\mathcal{F}_{\text{PAKE}}$" cannot be omitted, since otherwise (3) can be trivially satisfied by having no successful simulator.

[7] More precisely, assume w.l.o.g. that $\mathcal{S}$ sends $(\mathsf{NewKey}, sid, \mathsf{P}, \cdot)$ first and $(\mathsf{NewKey}, sid, \mathsf{P}', \cdot)$ next. Then when $\mathcal{S}$ sends $(\mathsf{NewKey}, sid, \mathsf{P}, \cdot)$, $\mathcal{F}_{\text{PAKE}}$ enters the third case, so P receives and outputs (sid, K) for $K \leftarrow \{0,1\}^\lambda$; when $\mathcal{S}$ sends $(\mathsf{NewKey}, sid, \mathsf{P}, \cdot)$, $\mathcal{F}_{\text{PAKE}}$ enters the second case, so P receives and outputs (sid, K') for $K' := K$.

It follows that

$$\Pr[\mathcal{Z} \text{ outputs 1 in the real world}] \geq 1 - \Pr[\mathsf{SpuriousGuess}(\mathcal{S}, \mathcal{Z})] - \Pr[\mathsf{NoOutput}(\mathcal{S}, \mathcal{Z})] - \mathbf{Dist}(\mathcal{S}, \mathcal{Z}),$$

i.e., in the real world P outputs K, P' outputs K', and $K = K'$ with overwhelming probability ($\mathbf{Dist}(\mathcal{S}, \mathcal{Z})$ is negligible since $\mathcal{S}$ is successful). This implies that Π is correct. ■

Discussion. In traditional game-based definitions, correctness and security are usually defined separately. In the UC framework, by contrast, there is usually one functionality achieving various desired properties, including correctness. However, as TrivialPAKE shows, correctness cannot be taken "for granted"; apart from the standard "sanity check" that UC-security implies game-based *security*, one should also try to prove (or disprove) that UC-security implies *correctness*.

Proposal 1 above can be viewed as simply "conceding" that in the context of PAKE, UC-security only implies a notion of security, and correctness needs to be separately defined — just as in game-based notions. On the other hand, Proposals 2 and 3 attempt to address the underlying reason that causes the issue, namely $\mathcal{F}_{\text{PAKE}}$ gives the ideal adversary too much power. Intuitively, TestPwd corresponds to an adversary incorporating a password guess pw^* in a message m^*, and replacing an honest party's message m with m^*. (The adversary sending random garbage is modeled as $\mathsf{pw}^* = \perp$.) Therefore, *any ideal adversary sending* TestPwd *should be viewed as modifying protocol messages, which disqualifies it from being a valid simulator for an eavesdropper*. Essentially, Proposals 2 and 3 are a formalization of this intuition.

4.2 Three Sets of Variants

In this section, we consider some variants of reasonable simulators (Definition 5).

Strong reasonable simulators. Requiring a simulator to be reasonable only implies that it cannot send TestPwd if the (real) adversary is an eavesdropper; this does not prevent the simulator from sending TestPwd *before* the adversary modifies a protocol message (if the adversary does modify one eventually). For example, suppose the adversary passes the first two protocol messages without modification, and modifies the third; a reasonable simulator may send TestPwd when the first message is sent. Intuitively, sending TestPwd "in advance" should not be considered reasonable, since this would again cause a discrepancy between the simulator's power and the adversary's power: the simulator modifies protocol messages even when the adversary does not.

We now consider the notion of *strong* reasonable simulators, which essentially says that the simulator is not allowed to send TestPwd unless *and until* the adversary modifies a protocol message. It turns out that formalizing this notion is not as straightforward as formalizing reasonable simulators: in the latter case we can consider the adversary and the simulator separately, whereas here we need to consider the *order* of the two parties' actions, namely the adversary

modifying a protocol message and the simulator sending TestPwd. Furthermore, these two events occur in two different worlds, so we cannot compare their timing directly.

We overcome this difficulty by requiring that the simulator send TestPwd only *with permission from the environment*, which bridges the real world and the ideal world since the environment participates in both worlds. Formally, assume w.l.o.g. that for each session *sid*, the environment always sends a (Modify, sid) message while instructing the adversary to modify a protocol message for the first time in this session, and does not send such messages anywhere else (so $\mathcal{Z}$ sends at most one Modify message for each session).[8] For any environment $\mathcal{Z}$, define $\mathsf{GeneralSpuriousGuess}(\mathcal{S}, \mathcal{Z})$ as the following event: both P and P′ are honest, and there exists a session *sid* in which $\mathcal{S}$ sends a $(\mathsf{TestPwd}, sid, \cdot, \cdot)$ message to $\mathcal{F}_{\mathrm{PAKE}}$ without receiving (Modify, sid) from $\mathcal{Z}$.

Definition 6. *We say a simulator $\mathcal{S}$ is* strong reasonable *if for any efficient environment $\mathcal{Z}$, $\Pr[\mathsf{GeneralSpuriousGuess}(\mathcal{S}, \mathcal{Z})]$ and $\Pr[\mathsf{NoOutput}(\mathcal{S}, \mathcal{Z})]$ are both negligible, where the probability is taken over the randomness generated in the execution of Π, as well as the random tapes of $\mathcal{S}$, $\mathcal{Z}$, and $\mathcal{F}_{PAKE}$.*

The following straightforward lemma states that the strong reasonability requirement is indeed stronger than the ordinary reasonability requirement:

Lemma 2. *Any strong reasonable simulator is also a reasonable simulator.*

Proof. Suppose $\mathcal{S}$ is a strong reasonable simulator. For any efficient environment $\mathcal{Z}$, let SID be the set of sessions in which $\mathcal{Z}$ instructs the adversary to be an eavesdropper. If $\mathsf{SpuriousGuess}(\mathcal{S}, \mathcal{Z})$ occurs, there exists a $sid \in SID$ such that $\mathcal{S}$ sends $(\mathsf{TestPwd}, sid, \cdot, \cdot)$ to $\mathcal{F}_{\mathrm{PAKE}}$. However, since the adversary never modifies a protocol message in *sid*, $\mathcal{Z}$ never sends a (Modify, sid) message, so $\mathsf{GeneralSpuriousGuess}(\mathcal{S}, \mathcal{Z})$ occurs. It follows that

$$\Pr[\mathsf{SpuriousGuess}(\mathcal{S}, \mathcal{Z})] \leq \Pr[\mathsf{GeneralSpuriousGuess}(\mathcal{S}, \mathcal{Z})],$$

which is negligible. So $\mathcal{S}$ is a reasonable simulator. ■

We now show that the notion of "reasonably realizing $\mathcal{F}_{\mathrm{PAKE}}$" remains equivalent if we replace reasonable simulators with strong reasonable simulators:

Lemma 3. *Let Π be any PAKE protocol. Then the followings are equivalent:*

(2⁺) There exists a successful strong reasonable simulator for Π;
(3) Π realizes $\mathcal{F}_{PAKE}$, and all successful simulators for Π are reasonable;

[8] This is w.l.o.g. because any environment $\mathcal{Z}$ can be converted into another environment $\mathcal{Z}'$ that behaves exactly like $\mathcal{Z}$, except that $\mathcal{Z}'$ additionally sends (Modify, sid) when $\mathcal{Z}$ instructs the adversary to modify a protocol message for the first time in session *sid*. Obviously the distinguishing advantages of $\mathcal{Z}$ and $\mathcal{Z}'$ are equal.

(3$^+$) Π *realizes* $\mathcal{F}_{PAKE}$*, and all successful simulators for* Π *are strong reasonable.*

Proof. *(2$^+$)⇒(3):* This is because (2$^+$) implies (2) by Lemma 2, and (2) implies (3) by Lemma 1.

(3)⇒(3$^+$): The high-level idea is that, before the adversary modifies a protocol message, a reasonable simulator does not know whether the adversary will eventually be an eavesdropper or not, so it "dare not" send TestPwd to $\mathcal{F}_{\text{PAKE}}$ (in case the adversary turns out to be an eavesdropper).

Let $\mathcal{S}$ be a successful simulator for Π; then $\mathcal{S}$ is reasonable, and we need to show that $\mathcal{S}$ is strong reasonable. Let $\mathcal{Z}$ be an efficient environment. As a warm-up, we first prove the lemma in the case that $\mathcal{Z}$ only initiates a single session *sid*. If $\mathcal{Z}$ instructs its adversary $\mathcal{A}$ to be an eavesdropper, then $\mathsf{SpuriousGuess}(\mathcal{S}, \mathcal{Z})$ and $\mathsf{GeneralSpuriousGuess}(\mathcal{S}, \mathcal{Z})$ are equivalent, so the lemma is immediate. Otherwise consider the following environment $\mathcal{Z}'$, which inputs the passwords that $\mathcal{Z}$ inputs, and instructs its adversary $\mathcal{A}'$ to be an eavesdropper:

Environment $\mathcal{Z}'$:

1. Run $\mathcal{Z}$. When $\mathcal{Z}$ sends $(\mathsf{NewSession}, sid, \mathsf{P}, \mathsf{P}', \mathsf{pw}, \mathsf{role})$ (resp. $(\mathsf{NewSession}, sid, \mathsf{P}', \mathsf{P}, \mathsf{pw}', \mathsf{role}')$) to P (resp. P'), send the same message to P (resp. P').
2. Instruct $\mathcal{A}'$ to be an eavesdropper in session *sid*.
3. When session is completed, output $b \leftarrow \{0,1\}$.[9]

Since $\mathcal{A}$ (the adversary corresponding to $\mathcal{Z}$) is not an eavesdropper, there exists an $r \in \mathbb{N}^+$ such that $\mathcal{A}$ does not modify the first $r-1$ protocol messages, but modifies the r-th protocol message. [10] The key observation is that $\mathcal{Z}$ *and* $\mathcal{Z}'$ *behave identically up to the* r*-th protocol message*, since both use password pw for P and password pw′ for P′, and both instruct the adversary to pass the first $r-1$ messages without modification. Therefore, we have (below we abbreviate $(\mathsf{TestPwd}, sid, \cdot, \cdot)$ as TestPwd):

$$
\begin{aligned}
&\Pr[\mathcal{S} \text{ sends } \mathsf{TestPwd} \text{ before receiving the } r\text{-th protocol message in the world of } \mathcal{Z}] \\
=&\Pr[\mathcal{S} \text{ sends } \mathsf{TestPwd} \text{ before receiving the } r\text{-th protocol message in the world of } \mathcal{Z}'] \\
\leq&\Pr[\mathcal{S} \text{ sends } \mathsf{TestPwd} \text{ in the world of } \mathcal{Z}'] \\
=&\Pr[\mathsf{SpuriousGuess}(\mathcal{S}, \mathcal{Z}')].
\end{aligned}
$$

However, since the r-th protocol message is the first time when $\mathcal{Z}$ instructs $\mathcal{A}$ to modify a protocol message, this is also when $\mathcal{Z}$ sends (Modify, sid). Therefore, (in

[9] In fact $\mathcal{Z}'$ does not need to output anything, since we rely on the fact that $\Pr[\mathsf{SpuriousGuess}(\mathcal{S}, \mathcal{Z}')]$ is negligible, rather than the distinguishing advantage of $\mathcal{Z}'$ is negligible; whether $\mathsf{SpuriousGuess}(\mathcal{S}, \mathcal{Z}')$ happens or not is already determined before $\mathcal{Z}'$ finally outputs a bit.

[10] Formally, r is a random variable depending on the random tape of $\mathcal{Z}$, and all probabilities below are also taken over the random tape of $\mathcal{Z}$.

the world of $\mathcal{Z}$) $\mathcal{S}$ receives (Modify, sid) together with the r-th protocol message from $\mathcal{Z}$. $\mathsf{GeneralSpuriousGuess}(\mathcal{S}, \mathcal{Z})$ is defined as $\mathcal{S}$ sends $\mathsf{TestPwd}$ before this, i.e., $\mathcal{S}$ sends $\mathsf{TestPwd}$ before receiving the r-th protocol message. So,

$$\begin{aligned}&\Pr[\mathcal{S} \text{ sends } \mathsf{TestPwd} \text{ before receiving the } r\text{-th protocol message in the world of } \mathcal{Z}]\\ =\ &\Pr[\mathsf{GeneralSpuriousGuess}(\mathcal{S}, \mathcal{Z})].\end{aligned}$$

Combining the above, we obtain

$$\Pr[\mathsf{GeneralSpuriousGuess}(\mathcal{S}, \mathcal{Z})] \leq \Pr[\mathsf{SpuriousGuess}(\mathcal{S}, \mathcal{Z}')],$$

which is negligible. This shows that $\mathcal{S}$ is strong reasonable.

In the general case, assume $\mathcal{Z}$ initiates q sessions $sid_1, \ldots, sid_q$. $\mathcal{Z}$ may instruct $\mathcal{A}$ to be an eavesdropper in some of them, and to modify messages in others. For $\ell \in [q]$, if $\mathcal{A}$ is not an eavesdropper, then there exist an $r_\ell \in \mathbb{N}^+$ such that $\mathcal{A}$ does not modify the first $r_\ell - 1$ protocol messages, but modifies the r_ℓ-th protocol message; if $\mathcal{A}$ is an eavesdropper, then let $r_\ell = \infty$. Environment $\mathcal{Z}'$ works exactly as in the simple-session case, except that it does not output the bit b until all sessions are completed. Just as the single-session argument above, we observe that *in session sid_ℓ, $\mathcal{Z}$ and $\mathcal{Z}'$ behave identically up to the r_ℓ-th protocol message*, so

$$\begin{aligned}&\Pr[\mathsf{GeneralSpuriousGuess}(\mathcal{S}, \mathcal{Z})]\\ =\ &\Pr[\text{There exists } \ell \in [q] \text{ such that } \mathcal{S} \text{ sends } (\mathsf{TestPwd}, sid_\ell, \cdot, \cdot)\\ &\qquad\qquad \text{before receiving the } r_\ell\text{-th protocol message in the world of } \mathcal{Z}]\\ \leq\ &\Pr[\mathsf{SpuriousGuess}(\mathcal{S}, \mathcal{Z}')],\end{aligned}$$

which is negligible. So $\mathcal{S}$ is strong reasonable.

($\exists^+$)$\Rightarrow$($\exists^+$): This is immediate. ■

(Strong) perfectly reasonable simulators. If we require that a simulator never make a spurious guess (rather than making it with negligible probability), we get:

Definition 7. *We say a simulator $\mathcal{S}$ is* perfectly reasonable *if for any efficient environment $\mathcal{Z}$, $\Pr[\mathsf{SpuriousGuess}(\mathcal{S}, \mathcal{Z})] = 0$ and $\Pr[\mathsf{NoOutput}(\mathcal{S}, \mathcal{Z})] = 0$, where the probability is taken over the randomness generated in the execution of Π, as well as the random tapes of $\mathcal{S}$, $\mathcal{Z}$, and $\mathcal{F}_{PAKE}$.*

Similarly, for general spurious guesses,

Definition 8. *We say a simulator $\mathcal{S}$ is* strong perfectly reasonable *if for any efficient environment $\mathcal{Z}$, $\Pr[\mathsf{GeneralSpuriousGuess}(\mathcal{S}, \mathcal{Z})] = 0$ and $\Pr[\mathsf{NoOutput}(\mathcal{S}, \mathcal{Z})] = 0$, where the probability is taken over the randomness generated in the execution of Π, as well as the random tapes of $\mathcal{S}$, $\mathcal{Z}$, and $\mathcal{F}_{PAKE}$.*

Remark 2. Interestingly, strong perfectly reasonable simulators have been studied while addressing a separate definitional issue in the context of *asymmetric* PAKE (aPAKE), where one party (called the user) holds the plain password and another party (called the server) holds a *password file* file, namely a one-way mapping of the password. For the purpose of our discussion, it suffices to let $\mathsf{file} = H(\mathsf{pw})$, where H is a random oracle. We want to ensure that after compromising the server and learning file, the adversary needs $\Theta(|\mathsf{Dict}|)$ time to recover pw.

Formally, this is modeled as follows: after compromising the server, the ideal adversary can send an $(\mathsf{OfflineTestPwd}, sid, \mathsf{pw}^*)$ command to the functionality, which returns "correct guess" (if pw^* is the correct password) or "wrong guess". Here an issue analogous to our "general spurious guess" emerges: what prevents the simulator from sending $\mathsf{OfflineTestPwd}$ for all $x \in \mathsf{Dict}$ and learning the password, before the real adversary makes any H queries?

This issue has been discussed in [15,20], where the proposed solution is similar to requiring the simulator to be strong perfectly reasonable; that is, the simulator is not allowed to send $(\mathsf{OfflineTestPwd}, sid, x)$, unless and until the real adversary queries $H(x)$. Both of the aforementioned works require $\mathsf{OfflineTestPwd}$ messages to be "accounted for by the environment" (yet neither of them formally defines what "accounted for by the environment" means).

However, it was later pointed out [19] that such a solution is insufficient for aPAKE. In a nutshell, this is because the random oracle H can be queried by the *environment* directly (rather than the environment instructing the real-world adversary to do so); therefore, an environment can learn H input/output pairs without sending any permission to the adversary, causing the simulator not being able to send any $\mathsf{OfflineTestPwd}$ messages (even when the environment already learns the password). (For a formal treatment of the discussion above, see [19, Appendix D].) We do not suffer from this issue, because in our setting all protocol messages must be passed by the real adversary, rather than the environment itself — in contrast to the environment being able to query the random oracle on its own.

It is not hard to see that for (2) and (2^+), the analogous conditions — where the simulator is required to be perfect — are equivalent to (2^+), whereas for (3) and (3^+), the analogous conditions cannot be satisfied:

Lemma 4. *Let* Π *be any PAKE protocol. Then the followings are equivalent:*

(2$^+$) There exists a successful strong reasonable simulator for Π*;*
(2) There exists a successful perfectly reasonable simulator for* Π*;*
(2^{*+}) There exists a successful strong perfectly reasonable simulator for Π*.*

Furthermore, the followings do not hold:

(3)* Π *realizes* $\mathcal{F}_{PAKE}$*, and all successful simulators for* Π *are perfectly reasonable;*
(3^{*+}) Π *realizes* $\mathcal{F}_{PAKE}$*, and all successful simulators for* Π *are strong perfectly reasonable.*

Proof. *(2$^+$)⇒(2^{*+}):* The intuition is that if and when GeneralSpuriousGuess or NoOutput occurs, the simulator can simply abort instead. Formally, let $\mathcal{S}$ be a successful strong reasonable simulator for Π. Consider the following simulator $\mathcal{S}'$:

Simulator $\mathcal{S}'$:

1. Upon receiving the first message from $\mathcal{F}_{\text{PAKE}}$, activate $\mathcal{S}$ using the same message. After that, pass messages between $\mathcal{S}$ and $\mathcal{F}_{\text{PAKE}}$ without any modifications. That is, upon receiving a message from $\mathcal{S}$ (aimed at $\mathcal{F}_{\text{PAKE}}$), send the same message to $\mathcal{F}_{\text{PAKE}}$; upon receiving a message from $\mathcal{F}_{\text{PAKE}}$, send the same message to $\mathcal{S}$ (as a message from $\mathcal{F}_{\text{PAKE}}$). Also, pass messages between $\mathcal{S}$ and $\mathcal{Z}$ without any modifications.
2. If at any time there exists a sid such that $\mathcal{S}'$ has not sent (Modify, sid) to $\mathcal{S}$, but $\mathcal{S}$ sends $(\mathsf{TestPwd}, sid, \mathsf{P}, \cdot)$ to $\mathcal{S}'$ (aimed at $\mathcal{F}_{\text{PAKE}}$), then send $(\mathsf{NewKey}, sid, \mathsf{P}, 0^\lambda)$ to $\mathcal{F}_{\text{PAKE}}$, output Abort and halt.
3. If there exists a sid such that $\mathcal{S}'$ has sent $(\mathsf{NewSession}, sid, \mathsf{P}, \cdot, \cdot)$ to $\mathcal{S}$, but $\mathcal{S}'$ does not send $(\mathsf{NewKey}, sid, \mathsf{P}, K \in \{0,1\}^\lambda)$ when it halts, then send $(\mathsf{NewKey}, sid, \mathsf{P}, 0^\lambda)$ to $\mathcal{F}_{\text{PAKE}}$, output Abort and halt.

Clearly, $\mathcal{S}'$ simulates the ideal game to $\mathcal{S}$ perfectly, and behaves exactly as $\mathcal{S}$ in its own ideal game, unless and until $\mathsf{GeneralSpuriousGuess}(\mathcal{S}, \mathcal{Z})$ or $\mathsf{NoOutput}(\mathcal{S}, \mathcal{Z})$ occurs (in which case $\mathcal{S}'$ sends NewKey to $\mathcal{F}_{\text{PAKE}}$ and outputs Abort). This means that $\mathsf{GeneralSpuriousGuess}(\mathcal{S}', \mathcal{Z})$ or $\mathsf{NoOutput}(\mathcal{S}', \mathcal{Z})$ never occurs, i.e., $\mathcal{S}'$ is strong perfectly reasonable; furthermore,

$$|\mathbf{Dist}(\mathcal{S}', \mathcal{Z}) - \mathbf{Dist}(\mathcal{S}, \mathcal{Z})| \leq \Pr[\mathsf{Abort}] = \Pr[\mathsf{GeneralSpuriousGuess}(\mathcal{S}, \mathcal{Z})] + \Pr[\mathsf{NoOutput}(\mathcal{S}, \mathcal{Z})],$$

so $\mathbf{Dist}(\mathcal{S}', \mathcal{Z}) \leq \mathbf{Dist}(\mathcal{S}, \mathcal{Z}) + \Pr[\mathsf{GeneralSpuriousGuess}(\mathcal{S}, \mathcal{Z})] + \Pr[\mathsf{NoOutput}(\mathcal{S}, \mathcal{Z})]$. Since $\mathcal{S}$ is successful, $\mathbf{Dist}(\mathcal{S}, \mathcal{Z})$ is negligible; since $\mathcal{S}$ is strong reasonable, $\Pr[\mathsf{GeneralSpuriousGuess}(\mathcal{S}, \mathcal{Z})]$ and $\Pr[\mathsf{NoOutput}(\mathcal{S}, \mathcal{Z})]$ are both negligible. We conclude that $\mathbf{Dist}(\mathcal{S}', \mathcal{Z})$ is negligible, i.e., $\mathcal{S}'$ is successful. This completes the proof.

(2^{*+})⇒(2*): This is immediate.

(2*)⇒(2$^+$): This is because (2*) trivially implies (2), (2) implies (3) by Lemma 1, and (3) implies (2$^+$) by Lemma 3.

(3) does not hold:* This is because a simulator can always send TestPwd with negligible probability (which does not affect it being successful). Formally, let $\mathcal{S}$ be a successful reasonable simulator for Π. Consider the following simulator $\mathcal{S}'$:

Simulator $\mathcal{S}'$:

1. On $(\mathsf{NewSession}, sid, \mathsf{P}, \mathsf{P}', \mathsf{role})$ from $\mathcal{F}_{\text{PAKE}}$, send $(\mathsf{TestPwd}, sid, \mathsf{P}, \perp)$ with probability $1/2^\lambda$. Then output Abort and halt.

2. If Abort does not occur, do the following: activate $\mathcal{S}$ with $(\mathsf{NewSession}, sid, \mathsf{P}, \mathsf{P}', \mathsf{role})$. After that, pass messages between $\mathcal{S}$ and $\mathcal{F}_{\mathrm{PAKE}}$ without any modifications, and also pass messages between $\mathcal{S}$ and $\mathcal{Z}$ without any modifications.

Let $\mathcal{Z}$ be the environment that runs a single session and instructs the adversary to be an eavesdropper. If Abort occurs, then $\mathcal{S}'$ sends TestPwd command in step 1, so $\mathsf{SpuriousGuess}(\mathcal{S}', \mathcal{Z})$ occurs. We have that

$$\Pr[\mathsf{SpuriousGuess}(\mathcal{S}', \mathcal{Z})] \geq \Pr[\mathsf{Abort}] = \frac{1}{2^\lambda},$$

i.e., $\mathcal{S}'$ is not perfectly reasonable. On the other hand, if Abort does not occur, then $\mathcal{S}'$ behaves exactly as $\mathcal{S}$. Therefore,

$$\mathbf{Dist}(\mathcal{S}', \mathcal{Z}) \leq \Pr[\mathsf{Abort}] + \mathbf{Dist}(\mathcal{S}, \mathcal{Z}) = \frac{1}{2^\lambda} + \mathbf{Dist}(\mathcal{S}, \mathcal{Z}),$$

which is negligible ($\mathbf{Dist}(\mathcal{S}, \mathcal{Z})$ is negligible because $\mathcal{S}$ is successful). So $\mathcal{S}'$ is successful.

We have a simulator for Π that is successful but not perfectly reasonable, which contradicts (3*).

(3$^+$*) does not hold:* This is because (3*$^+$) is stronger than (3*), and we have just proved that (3*) does not hold. ■

Remark 3. The proof of (2$^+$)⇒(2*$^+$) critically relies on the fact that *once* $\mathsf{GeneralSpuriousGuess}(\mathcal{S}, \mathcal{Z})$ *occurs,* $\mathcal{S}'$ *can detect it before sending any* TestPwd *commands.* The same thing cannot be said for SpuriousGuess. That is, if we wanted to prove (2)⇒(2*) directly, i.e., $\mathcal{S}$ is only reasonable (but not necessarily strong reasonable), then $\mathcal{S}'$ would run into trouble: $\mathcal{S}'$ needs to wait until the end of a session to determine whether $\mathsf{SpuriousGuess}(\mathcal{S}, \mathcal{Z})$ occurred or not (recall that $\mathsf{SpuriousGuess}(\mathcal{S}, \mathcal{Z})$ requires the adversary to be an eavesdropper *throughout the session*), at which point $\mathcal{S}'$ might have already sent TestPwd, which might be a "spurious guess" if the adversary eventually turns out to be an eavesdropper. It seems that the only way to prove (2)⇒(2*) is via the chain (2)⇒(2$^+$)⇒(2*$^+$)⇒(2*).

4.3 Putting It Together

By Lemmas 1, 3 and 4, we get:

Theorem 1. *Let* Π *be any PAKE protocol. Then the followings are equivalent:*

(1) Π *is correct and realizes* $\mathcal{F}_{PAKE}$*;*
(2) There exists a successful reasonable simulator for Π*;*
(2) There exists a successful perfectly reasonable simulator for* Π*;*
(2$^+$) There exists a successful strong reasonable simulator for Π*;*

(2+) There exists a successful strong perfectly reasonable simulator for* Π*;*
(3) Π *realizes* $\mathcal{F}_{PAKE}$*, and all successful simulators for* Π *are reasonable;*
(3+) Π *realizes* $\mathcal{F}_{PAKE}$*, and all successful simulators for* Π *are strong reasonable.*

Furthermore, the followings do not hold:

(3)* Π *realizes* $\mathcal{F}_{PAKE}$*, and all successful simulators for* Π *are perfectly reasonable;*
(3+)* Π *realizes* $\mathcal{F}_{PAKE}$*, and all successful simulators for* Π *are strong perfectly reasonable.*

Since all seven conditions in the first part of Theorem 1 are equivalent, we can now define the notion of *reasonably realizing* the PAKE functionality:

Definition 9. *We say a PAKE protocol* Π $\mathsf{reasonably\ realizes}$ $\mathcal{F}_{\mathrm{PAKE}}$ *if* Π *is correct and realizes* $\mathcal{F}_{PAKE}$*, i.e.,* Π *satisfies condition (1) in Theorem 1 (equivalently,* Π *satisfies any of conditions (2), (2*), (2+), (2*+), (3), and (3+) in Theorem 1).*

5 Impossibility of a Direct Solution

All proposals in Section 4 are somewhat unsatisfactory, in that they either require correctness to be separate from security (Proposal 1) or place additional requirements on the UC simulator, hence changing the very definition of UC-security (all other proposals). Is it possible to have a direct solution, i.e., to incorporate correctness directly into the UC *functionality*, without changing the definition of UC-security? In this section, we give a negative answer.

Theorem 2. *There does not exist a UC functionality* $\mathcal{F}$ *such that a PAKE protocol* Π *reasonably realizes* $\mathcal{F}_{PAKE}$ *if and only if it realizes* $\mathcal{F}$*.*

Proof. The high-level idea is as follows: in a PAKE protocol, if the protocol messages are $\perp$, then the two parties output independent random keys. This means that there must be some mechanism in $\mathcal{F}$ that allows the two parties to output independent random keys. But then a simulator can use the same mechanism to complete the simulation for TrivialPAKE. The formal proof follows.

Assume towards contradiction that there exists such a UC functionality $\mathcal{F}$. Take any "natural" PAKE protocol that reasonably realizes $\mathcal{F}_{\mathrm{PAKE}}$; for concreteness, here we use Diffie-Hellman-based Encrypted Key Exchange (DH-EKE, Figure 4). Correctness can be checked as follows: assuming both parties P and P' hold the same password pw, and the adversary is an eavesdropper, then party P outputs

$$H((\mathcal{D}_{\mathsf{pw}}(Y))^x) = H((\mathcal{D}_{\mathsf{pw}}(\mathcal{E}_{\mathsf{pw}}(g^y)))^x) = H((g^y)^x) = H(g^{xy})$$

The protocol uses a group $(\mathbb{G}, g, q)$, an ideal cipher $(\mathcal{E}, \mathcal{D})$ where $\mathcal{E} : \mathsf{Dict} \times \mathbb{G} \to \{0,1\}^\lambda$ and $\mathcal{D} : \mathsf{Dict} \times \{0,1\}^\lambda \to \mathbb{G}$, and a random oracle $H : \mathbb{G} \to \{0,1\}^\lambda$. The protocol is completely symmetric, so we only describe the behavior of party P.

1. On input $(\mathsf{NewSession}, sid, \mathsf{P}, \mathsf{P}', \mathsf{pw}, \mathsf{role})$, if this is the first $\mathsf{NewSession}$ message for sid, party P samples $x \leftarrow \mathbb{Z}_q$, computes $X := \mathcal{E}_{\mathsf{pw}}(g^x)$, and sends (sid, X) to P'.
2. On (sid, Y) from party P', if $Y \notin \{0,1\}^\lambda$, then party P samples $K \leftarrow \{0,1\}^\lambda$ and outputs (sid, K). Otherwise P computes $K := H(\mathcal{D}_{\mathsf{pw}}(Y))$ and outputs (sid, K).

Fig. 4: Protocol DH-EKE

together with sid, and so does P'. Furthermore, it has been proven that DH-EKE realizes $\mathcal{F}_{\text{PAKE}}$ in the ideal cipher model and the random oracle model, under the computational Diffie-Hellman assumption in the group $(\mathbb{G}, g, q)$ [14, 24]. Therefore, DH-EKE reasonably realizes $\mathcal{F}_{\text{PAKE}}$ and thus realizes $\mathcal{F}$.

Let $\mathcal{S}$ be the simulator for DH-EKE realizing $\mathcal{F}$. We now use $\mathcal{S}$ to show that TrivialPAKE also realizes $\mathcal{F}$. The simulator $\mathcal{S}'$ works as follows:

Simulator $\mathcal{S}'$:

1. Upon receiving the first message from $\mathcal{F}$, activate $\mathcal{S}$ using the same message. After that, pass messages between $\mathcal{S}$ and $\mathcal{F}$ without any modifications. That is, upon receiving a message from $\mathcal{S}$ (aimed at $\mathcal{F}$), send the same message to $\mathcal{F}$; upon receiving a message from $\mathcal{F}$, send the same message to $\mathcal{S}$ (as a message from $\mathcal{F}$).
2. Upon receiving a message (sid, X) from $\mathcal{S}$ as a message from P to P', send $(sid, \perp)$ to $\mathcal{S}$ as a message from P' to P. The same goes for P'.
3. Continue passing messages between $\mathcal{S}$ and $\mathcal{F}$.

We now prove that $\mathcal{S}'$ is successful. As warm-up, we first show that if $\mathcal{F} = \mathcal{F}_{\text{PAKE}}$, then $\mathcal{S}'$ is exactly the simulator for TrivialPAKE in the proof of Proposition 1. The first message $\mathcal{S}'$ receives from $\mathcal{F}_{\text{PAKE}}$ is $(\mathsf{NewSession}, sid, \mathsf{P}, \mathsf{P}', \mathsf{role})$, and $\mathcal{S}'$ passes this message to $\mathcal{S}$. Then $\mathcal{S}$ begins to simulate protocol messages, i.e., $\mathcal{S}$ sends (sid, X) aimed at P'. $\mathcal{S}'$ then behaves like an environment that instructs the adversary to respond with $(sid, \perp)$. Upon receiving $(sid, \perp)$, $\mathcal{S}$ needs to simulate P's behavior of outputting a random key; it does so by sending $(\mathsf{TestPwd}, sid, \mathsf{P}, \perp)$ followed by $(\mathsf{NewKey}, sid, \mathsf{P}, 0^\lambda)$ to $\mathcal{F}_{\text{PAKE}}$ (whose role is played by $\mathcal{S}'$).[11] This means that $\mathcal{S}'$ also sends these two messages to its own $\mathcal{F}_{\text{PAKE}}$. At this point we recover the simulator in the proof of Proposition 1.

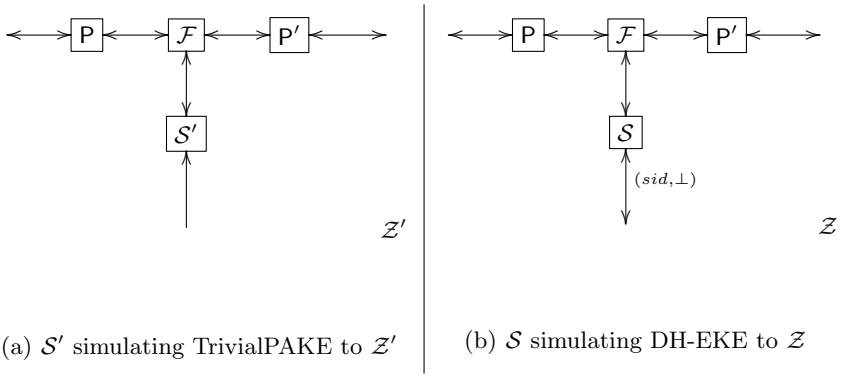
[11] 0^λ can be replaced by any string in $\{0,1\}^\lambda$.

In general, the crucial point is that $\mathcal{S}'$, while communicating with $\mathcal{S}$, behaves like an environment that instructs the real adversary to send $(sid, \perp)$ to protocol parties (causing them to output independent random keys). To see this formally, for any environment $\mathcal{Z}'$ in the world of $\mathcal{S}'$ and attacking TrivialPAKE, consider the following environment $\mathcal{Z}$ in the world of $\mathcal{S}$ and attacking DH-EKE:

Environment $\mathcal{Z}$:

1. When $\mathcal{Z}'$ activates a new session for $\langle \mathsf{P}, \mathsf{P}' \rangle$ on password pw, do the same thing.
2. Instruct the adversary to send $(sid, \perp)$ as protocol messages.

Recall that $\mathcal{S}$ communicates with two parties: $\mathcal{F}$ to which it sends UC commands, and $\mathcal{Z}$ to which it sends protocol messages (as messages from P aimed at P', or vice versa). $\mathcal{S}'$ plays the roles of both $\mathcal{F}$ and $\mathcal{Z}$ to $\mathcal{S}$. For the former interface, $\mathcal{S}'$ merely passes all messages from $\mathcal{F}$ to $\mathcal{S}$, and from $\mathcal{S}$ to $\mathcal{F}$; furthermore, $\mathcal{Z}'$ behaves exactly like $\mathcal{Z}$ when sending messages to $\mathcal{F}$ via P. Therefore, the view of $\langle \mathcal{S}' \rightleftarrows \mathcal{F} \rightleftarrows \mathsf{P} \rightleftarrows \mathcal{Z}' \rangle$ is identical to the view of $\langle \mathcal{S} \rightleftarrows \mathcal{F} \rightleftarrows \mathsf{P} \rightleftarrows \mathcal{Z} \rangle$. On the other hand, $\mathcal{S}'$ replaces all protocol messages with $(sid, \perp)$, which is exactly what $\mathcal{Z}$ does. We conclude that $\mathcal{S}'$ perfectly simulates $\mathcal{Z}$ to $\mathcal{S}$.



(a) $\mathcal{S}'$ simulating TrivialPAKE to $\mathcal{Z}'$

(b) $\mathcal{S}$ simulating DH-EKE to $\mathcal{Z}$

Fig. 5: Comparison of $\mathcal{S}'$ simulating TrivialPAKE to $\mathcal{Z}'$ and $\mathcal{S}$ simulating EKE to $\mathcal{Z}$. The messages in the upper halves of (a) and (b) are identical. However, in (b), $\mathcal{Z}$ (whose role is played by $\mathcal{S}'$ in (a)) sends $(sid, \perp)$ as protocol messages to $\mathcal{S}$. Since $\mathcal{S}$ is successful, P and P' must output independent random keys in (b), which in turn implies that P and P' output independent random keys in (a), so $\mathcal{S}$‘ is also successful. Note that $\mathcal{S}'$ never sends any messages to $\mathcal{Z}'$, as there are no protocol messages to simulate.

Since $\mathcal{Z}$ sends $(sid, \perp)$ as protocol messages, in a real execution, this causes P and P' to output independent random keys in $\{0,1\}^\lambda$ (together with sid).

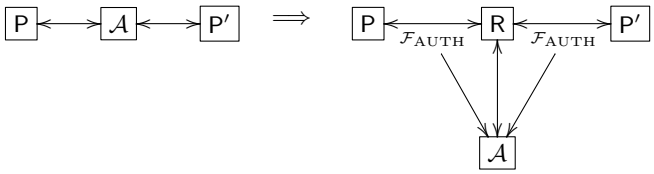


Fig. 6: Changes to the real-world model used by PAKE. The left is for the usual PAKE definition with a man-in-the-middle adversary, while the right shows the router model (where the router R is corrupted). The unauthenticated channel becomes two authenticated channels, with an extra router party passing messages back and forth.

Thus, in the ideal world simulated by $\mathcal{S}$, the output distribution of parties is indistinguishable with each party independently outputting a random string in $\{0,1\}^\lambda$ (together with *sid*) per session. Recall again that the messages sent between $\mathcal{S}'$ and $\mathcal{F}$ (whose role is played by $\mathcal{S}$) are exactly the same with the messages sent between $\mathcal{S}$ and $\mathcal{F}$; it follows that the parties' output distribution in the world of $\mathcal{Z}$ and $\mathcal{S}$ is also indistinguishable with the above, i.e., each party independently outputting a random string in $\{0,1\}^\lambda$ (together with *sid*) per session. But TrivialPAKE has no protocol messages, and the only strings $\mathcal{Z}$ receives are parties' outputs (also see the proof of Proposition 1). This shows that the view of $\mathcal{Z}$ simulated by $\mathcal{S}$ is indistinguishable from the real view — i.e., $\mathcal{S}$ is "as successful as" $\mathcal{S}'$. Therefore, TrivialPAKE realizes $\mathcal{F}$.

Since TrivialPAKE does not satisfy correctness, it does not reasonably realize $\mathcal{F}_{\text{PAKE}}$. So we have a PAKE protocol that does not reasonably realize $\mathcal{F}_{\text{PAKE}}$ but realizes $\mathcal{F}$, which contradicts the hypothesis about $\mathcal{F}$. ■

6 PAKE as a Three-Party Protocol

In this section, we show how to bypass the impossibility result in the previous section and directly incorporating correctness into the PAKE functionality, by changing the execution model of the PAKE to add a third party called the *router* R. Such a definition allows a UC PAKE to compose with other protocols in the normal sense of UC composition, allowing the higher level protocols to use the fact that the PAKE is correct in a natural way.

Concretely, any protocol in the router model is required to have a distinguished party R, and authenticated channels (Figure 7) connecting every party to R; parties other than R do not communicate with each other directly. The protocol's code for R must tell it to simply route messages between the parties — where each message sent to R is prefixed with the desired

– On input (sid, R, m) from S, send $(sid, \mathsf{S}, \mathsf{R}, m)$ to $\mathcal{A}$. Wait for (Ok, sid) from $\mathcal{A}$. Then send (sid, S, m) to R.

Fig. 7: The authenticated channel functionality, $\mathcal{F}_{\text{AUTH}}$ (includes highlighted), together with it's guaranteed delivery variant $\mathcal{F}'_{\text{AUTH}}$ (excludes highlighted). Note that we do not allow corrupted parties to modify their own messages – this feature is unimportant for our purposes. $\mathcal{F}'_{\text{AUTH}}$ is modified from $\mathcal{F}_{\text{AUTH}}$ to guarantee that every message is delivered eventually. This is needed in Section 6.2, as otherwise the protocol might not complete even when both parties are honest.

destination, while each message sent by R is prefixed with the original source. However, R is a corruptible party like any other, and when corrupted the adversary can have it modify the messages arbitrarily. Intuitively, R represents the man-in-the-middle adversary, and an honest R corresponds to the adversary being an eavesdropper. The modified structure for PAKE protocols in the router model is illustrated in Figure 6.

6.1 Correctness

In this section, we first deal with the case that both protocol parties must output a key. This covers TrivialPAKE but not the "empty" protocol. In Figure 8, we present a modified PAKE ideal functionality that works in the router model. Below we show that it is equivalent to a UC PAKE that satisfies correctness.

Definition 10. *A PAKE protocol* Π *has* guaranteed output when all parties are honest*, or simply* guaranteed output*, if in any session where all parties are honest (including the router* R*), the protocol parties both output a string in* $\{0,1\}^\lambda$ *(together with sid) with overwhelming probability before they halt.*

Theorem 3. *For any PAKE protocol* Π *that is based on an unauthenticated channel and has guaranteed output, there is a corresponding router model PAKE protocol* $\widetilde{\Pi}$ *that is based on an authenticated channel and has guaranteed output, and vice versa, such that the following conditions are equivalent:*

(a) Π ***reasonably*** *realizes* $\mathcal{F}_{PAKE}$ *(Definition 9);*
(b) $\widetilde{\Pi}$ *realizes* $\mathcal{F}_{PAKE\text{-}3}$ *(Figure 8), where* R *follows the* ***static*** *corruption model;*
(c) $\widetilde{\Pi}$ *realizes* $\mathcal{F}_{PAKE\text{-}3}$*, where* R *follows the* ***adaptive*** *corruption model.*

Proof. The correspondence between Π and $\widetilde{\Pi}$ is already completely specified by the router model. Below we prove equivalence of the three conditions.

(a)⇒(c): Let $\mathcal{S}$ be a successful strong perfectly reasonable simulator for Π, which must exist by Theorem 1. We now construct a simulator $\widetilde{\mathcal{S}}$ for $\widetilde{\Pi}$.

– On input (NewSession, sid, P, P′, pw, role) from P, create a record ⟨P, P′, pw⟩ and mark it fresh if either: (a) this is the first NewSession message for sid, or (b) this is the second NewSession message for sid and there is a record ⟨P′, P, ·⟩. Send (NewSession, sid, P, P′, role) to $\mathcal{S}$.

After sending NewSession, check if case (b) happened and P, P′ and R are all honest. If so, run the NewKey handler below for P and then P′, i.e., behave as if the messages (NewKey, sid, P, 0^λ) and (NewKey, sid, P′, 0^λ) were sent by $\mathcal{S}$.

– On (TestPwd, sid, P, pw*) from R or P′, if it is corrupted and there is a record ⟨P, P′, pw⟩ marked fresh, then do:
  - If pw* = pw, mark the record compromised and send "correct guess" to R.
  - If pw* ≠ pw, mark the record interrupted and send "wrong guess" to R.

– On (NewKey, sid, P, $K^* \in \{0,1\}^\lambda$) from $\mathcal{S}$, if there is a record ⟨P, P′, pw⟩, and this is the first NewKey message for sid and P, then output (sid, K) to P, where K is defined as follows:
  - If the record is compromised, or either P or P′ is corrupted, then set $K := K^*$.
  - If the record is fresh, and a key (sid, K') has been output to P′, at which time there was a record ⟨P′, P, pw⟩ marked fresh, then set $K := K'$.
  - Otherwise sample $K \leftarrow \{0,1\}^\lambda$.

  Finally, mark the record completed.

Fig. 8: Three-party model UC functionality $\mathcal{F}_{\text{PAKE-3}}$ for PAKE. Differences with the standard PAKE functionality (Figure 2) are highlighted in yellow. $\mathcal{F}'_{\text{PAKE-3}}$ adds the lines highlighted in red to guarantee output.

Essentially, $\widetilde{\mathcal{S}}$ does whatever $\mathcal{S}$ does, except that $\widetilde{\mathcal{S}}$ treats the corrupted R as the man-in-the-middle adversary while simulating protocol messages and sending TestPwd commands:

Simulator $\widetilde{\mathcal{S}}$:

1. Upon receiving the first message from $\mathcal{F}_{\text{PAKE-3}}$ (which must be NewSession), activate $\mathcal{S}$ using the same message. After that, forward $\mathcal{F}_{\text{PAKE-3}}$'s other NewSession messages to $\mathcal{S}$ (as messages from $\mathcal{F}_{\text{PAKE}}$).
2. On (sid, m) from $\mathcal{S}$ as the simulation of a protocol message from P to P′, if R is honest, then send (sid, P, R, m) and (sid, R, P′, m) to $\widetilde{\mathcal{Z}}$ (as messages from $\mathcal{F}_{\text{AUTH}}$ to the adversary; same below), and (sid, m) to $\mathcal{S}$ (as a message from $\mathcal{S}$'s environment; same below). If R is corrupted, then send (sid, P, R, m) to $\widetilde{\mathcal{Z}}$ and (sid, m) to R; on (sid, m') from R, send (sid, R, P′, m') to $\widetilde{\mathcal{Z}}$ and (sid, m') to $\mathcal{S}$.
3. On (TestPwd, sid, P, pw*) from $\mathcal{S}$, if either R or P′ is corrupted, then forward this message to $\mathcal{F}_{\text{PAKE-3}}$ (as a message from the corrupted party), and forward $\mathcal{F}_{\text{PAKE-3}}$'s response ("correct guess" or "wrong guess") back to $\mathcal{S}$.

On the other hand, if both R and P' are honest but P is corrupted, note that $\mathcal{F}_{\text{PAKE-3}}$ does not allow a party to attack itself — that is, P cannot trigger $(\mathsf{TestPwd}, sid, \mathsf{P}, \mathsf{pw}^*)$. Instead, simply check if pw^* matches the password of P, which can be seen from the $\mathsf{NewSession}$ message.

4. On $(\mathsf{NewKey}, sid, \cdot, \cdot)$ from $\mathcal{S}$, forward this message to $\mathcal{F}_{\text{PAKE-3}}$.

We now prove that $\widetilde{\mathcal{S}}$ is successful; that is, for any efficient environment $\widetilde{\mathcal{Z}}$ attacking $\widetilde{\Pi}$, $\widetilde{\mathcal{S}}$ generates a view indistinguishable from the real view of $\widetilde{\mathcal{Z}}$. The key point is that $\widetilde{\mathcal{S}}$ acts like an environment — whose behavior corresponds to $\widetilde{\mathcal{Z}}$'s behavior — in the view of $\mathcal{S}$. To formalize this argument, define an environment $\mathcal{Z}$ in the world of $\mathcal{S}$ and attacking Π:

Environment $\mathcal{Z}$:

1. When $\widetilde{\mathcal{Z}}$ activates a new session for $\langle \mathsf{P}, \mathsf{P}' \rangle$ on password pw, do the same thing.
2. As long as R is honest, instruct the adversary to transmit messages between P and P' without any modifications.
3. When R is corrupted, $\widetilde{\mathcal{Z}}$ may let R modify the protocol messages arbitrarily, and $\mathcal{Z}$ matches this by instructing its own adversary modify the unauthenticated channel messages in the same way. That is, when R receives (sid, m) from P and sends (sid, m') to P', instruct the adversary to send (sid, m') to P'.
4. When $\widetilde{\mathcal{Z}}$ outputs a bit b, output the same bit b.

In the real world, the only difference between an execution of $\widetilde{\Pi}$ by $\widetilde{\mathcal{Z}}$ and the corresponding execution of Π by $\mathcal{Z}$ is how the messages are passed and modified: in $\widetilde{\Pi}$, the protocol messages are passed through $\mathcal{F}_{\text{AUTH}}$ to the router, which may modify them when corrupted; while in Π, the protocol messages are passed and modified directly by the man-in-the-middle adversary. However, we made $\mathcal{Z}$ instruct the man-in-middle adversary to apply the same message modifications as would be made by the router (as can be seen in step 3 above). Therefore, the view of $\widetilde{\mathcal{Z}}$ in the execution of $\widetilde{\Pi}$ is identical to the view of $\mathcal{Z}$ in the corresponding execution of Π. We have that

$$\Pr[\widetilde{\mathcal{Z}} \text{ outputs 1 in the real world}] = \Pr[\mathcal{Z} \text{ outputs 1 in the real world}].$$

In the ideal world, we first claim that the view of $\mathcal{S}$ when run by $\widetilde{\mathcal{S}}$, is identical to the view of $\mathcal{S}$ in the world of $\mathcal{Z}$. Indeed, the behavior of $\widetilde{\mathcal{S}}$ when communicating with $\mathcal{S}$ is exactly the same as $\mathcal{Z}$ and $\mathcal{F}_{\text{PAKE}}$ combined, except that when P, P' and R are all honest, $\widetilde{\mathcal{S}}$ ignores $\mathsf{TestPwd}$ commands from $\mathcal{S}$. However, notice that as long as all parties are honest, $\mathcal{Z}$ instructs the adversary to transmit messages between P and P' without any modifications, so $\mathcal{S}$ sending $\mathsf{TestPwd}$ to $\mathcal{F}_{\text{PAKE}}$ means that $\mathsf{GeneralSpuriousGuess}(\mathcal{S}, \mathcal{Z})$ occurs — which violates the assumption that $\mathcal{S}$ is a strong perfectly reasonable simulator for Π. That is, it is impossible for $\mathcal{S}$ to send $\mathsf{TestPwd}$ when all parties are honest, so $\widetilde{\mathcal{S}}$ does not need to consider such an event.

We now argue that the view of $\widetilde{\mathcal{Z}}$ simulated by $\widetilde{\mathcal{S}}$ is identical to the view of $\mathcal{Z}$ simulated by $\mathcal{S}$. For protocol messages, $\widetilde{\mathcal{S}}$ simply sends whatever messages simulated by $\mathcal{S}$ to $\widetilde{\mathcal{Z}}$ (as from $\mathcal{F}_{\text{AUTH}}$). For the outputs from P and P', they are triggered by a NewKey command from $\widetilde{\mathcal{S}}$. $\widetilde{\mathcal{S}}$ sends TestPwd to $\mathcal{F}_{\text{PAKE-3}}$ whenever $\mathcal{S}$ sends such a message to $\mathcal{F}_{\text{PAKE}}$, and sends NewKey to $\mathcal{F}_{\text{PAKE-3}}$ whenever $\mathcal{S}$ sends such a message to $\mathcal{F}_{\text{PAKE}}$. Finally, note that $\mathcal{F}_{\text{PAKE-3}}$ processes NewKey messages in the same way as $\mathcal{F}_{\text{PAKE}}$ does. We conclude that

$$\Pr[\widetilde{\mathcal{Z}} \text{ outputs 1 in the ideal world with } \widetilde{\mathcal{S}}] = \Pr[\mathcal{Z} \text{ outputs 1 in the ideal world with } \mathcal{S}].$$

Combining the above, we get that $\mathbf{Dist}_{\widetilde{\Pi},\mathcal{F}_{\text{PAKE-3}}}(\widetilde{\mathcal{S}},\widetilde{\mathcal{Z}}) = \mathbf{Dist}_{\Pi,\mathcal{F}_{\text{PAKE}}}(\mathcal{S},\mathcal{Z})$, which is negligible since $\mathcal{S}$ is successful. This shows that $\widetilde{\mathcal{S}}$ is successful, i.e., $\widetilde{\Pi}$ realizes $\mathcal{F}_{\text{PAKE-3}}$.

(c)⇒(b): This implication is trivial, because the adaptive corruption model is stronger than the static corruption model.

(b)⇒(a): We first prove that Π realizes $\mathcal{F}_{\text{PAKE}}$. Let $\widetilde{\mathcal{S}}$ be a successful simulator for $\widetilde{\Pi}$. The simulator $\mathcal{S}$ for Π is relatively simple so we only provide a sketch: $\mathcal{S}$ runs $\widetilde{\mathcal{S}}$, passing its inputs and outputs to the appropriate parties and functionalities. While simulating the environment for $\widetilde{\mathcal{S}}$, since R does not exist for Π, $\mathcal{S}$ runs a fake party itself, treating R as always corrupted. Any communication between $\mathcal{S}$ and $\mathcal{F}_{\text{PAKE-3}}$ is passed through directly by $\widetilde{\mathcal{S}}$ (using $\mathcal{F}_{\text{PAKE}}$ instead of $\mathcal{F}_{\text{PAKE-3}}$). For protocol messages, $\mathcal{S}$ needs to translate between protocol message tampering by $\mathcal{A}$ and tampering by R — that is, when $\widetilde{\mathcal{S}}$ sends e.g., $(sid, \mathsf{P}, \mathsf{R}, m)$ as a message from $\mathcal{F}_{\text{AUTH}}$ to the adversary, $\mathcal{S}$ sends (sid, m) as a message from P to P'; when $\mathcal{S}$ receives (sid, m') aimed at P', it sends the same message to $\widetilde{\mathcal{S}}$ as a message from R.

To show that $\mathcal{S}$ is successful, for any environment $\mathcal{Z}$ attacking Π, let $\widetilde{\mathcal{Z}}$ be an environment attacking $\widetilde{\Pi}$ that corrupts R at the beginning of the protocol. Whenever $\mathcal{Z}$ instructs its adversary to tamper with protocol messages in Π, $\widetilde{\mathcal{Z}}$ translates this into tampering by R in $\widetilde{\Pi}$. With this change to the environment, the real and ideal worlds of Π exactly match the real and ideal worlds of $\widetilde{\Pi}$, and following the same structure as the proof of (a)⇒(c), we can see that $\mathcal{S}$ is successful.

Finally, we must prove that Π is correct. To see this, consider an environment $\widetilde{\mathcal{Z}}$ attacking $\widetilde{\Pi}$, which activates a session between P and P' with the two parties using the same password pw, and outputs 1 if P and P' output the same key when they halt. ($\widetilde{\mathcal{Z}}$ does not corrupt R.) In the ideal world of $\widetilde{\mathcal{Z}}$, TestPwd cannot be used so session records must remain fresh, guaranteeing that P and P' output the same key. This means that in a real execution of $\widetilde{\Pi}$, when P and P' are honest using the same password pw, and R is honest, P and P' must eventually output the same key with overwhelming probability. (The argument above relies on the fact that $\widetilde{\Pi}$ has guaranteed output, i.e., P and P' must output *some* key in $\widetilde{\Pi}$.) By the definition of $\widetilde{\Pi}$, this immediately implies the correctness of Π, where the condition "R is honest" is replaced by "the adversary is an eavesdropper". ■

6.2 PAKE Guaranteeing Output

Figure 6 also presents $\mathcal{F}'_{\text{PAKE-3}}$, another PAKE ideal functionality in the router model, this time designed to guarantee output as well as correctness. The idea is to have the ideal functionality itself trigger the NewKey interface if the simulator does not do it.

Unfortunately, two complications come with this change to the router model PAKE functionality. First, we need to guarantee delivery of all messages sent through the authenticated channel, as otherwise this functionality is impossible to realize. We use a modified authenticated channel functionality $\mathcal{F}'_{\text{AUTH}}$ (see Figure 7), where the message is both sent to the adversary and delivered directly.

Parallel execution. More importantly, these modified ideal functionalities now send more than one message per activation, which is unusual for UC. Normally ideal functionalities (as well as adversaries and simulators) are supposed to send just one message and halt, not send other messages that might be processed in parallel — in which case there might a "race condition".

To see why this is problematic, consider the following example: let the environment start a PAKE session with P, P′, and R all honest. The corresponding protocol messages must then be simulated. If the ideal functionality were to "run faster" than the simulator, i.e., if it triggers the NewKey handlers before the simulator finishes generating the protocol messages, then the key will be delivered too early. This would let the real world and ideal world be easily distinguished.

We clarify the execution order by specifying that if program is of the form "Send message m to party X, then do y", the action y waits until after m is sent and every action that occurs as a result of send m completes.[12] For example, with adaptive corruption, if the adversary receives $(sid, \mathsf{P}, \mathsf{R}, m)$ from $\mathcal{F}'_{\text{AUTH}}$ it could decide to corrupt R; this corruption would take place *before* (sid, S, m) is sent to R. This ensures that there is still only one thread of execution occurring at a time, and the ideal functionality merely writes down some action for later (when there is nothing left to do).

While this is not a usual description of UC functionalities, it can be made rigorous using the techniques from [21]. In their model, the environment is given the power to slowly advance the protocol through the honest party's interfaces, by repeatedly triggering an Output interface in the functionality. On every Output message[13], let the ideal functionality start performing whatever action was saved for later. As long as these queries are made by the environment, the functionality will work as described above.

[12] This can be viewed as a priority system. In programs of this form, we assign the action y a lower priority than sending m to X, all processing done by X, all messages sent by X as a result, and so on.

[13] Technically, Output must be sent by every honest party. Additionally, it must be sent some polynomial number of times, not just once, to allow protocols with multiple rounds. See [21] for details.

Theorem 4. *A router model PAKE protocol $\widetilde{\Pi}$ realizes $\mathcal{F}_{PAKE\text{-}3}$ with guaranteed output if and only if $\widetilde{\Pi}$ realizes $\mathcal{F}'_{PAKE\text{-}3}$ (in the guaranteed delivery router model, i.e., all protocol messages are sent via $\mathcal{F}'_{AUTH}$). This holds whether the router is modeled with static or adaptive corruption.*

Proof. *$\mathcal{F}'_{PAKE\text{-}3} \Rightarrow \mathcal{F}_{PAKE\text{-}3}$ with guaranteed output:* It is trivial that $\widetilde{\Pi}$ realizes $\mathcal{F}_{\text{PAKE-3}}$, because removing the extra NewKey trigger in $\mathsf{NewSession}$ only strengthens the simulator's power, as does adding the ability to not deliver messages sent through $\mathcal{F}_{\text{AUTH}}$. That is, the simulator can send $(\mathsf{NewKey}, sid, \mathsf{P}, 0^\lambda)$ and $(\mathsf{NewKey}, sid, \mathsf{P}', 0^\lambda)$ to $\mathcal{F}_{\text{PAKE-3}}$ at the end, like $\mathcal{F}'_{\text{PAKE-3}}$ would have done, and can always choose to have all messages delivered eventually, like in $\mathcal{F}'_{\text{AUTH}}$.

We now show that $\widetilde{\Pi}$ has guaranteed output. Consider the following environment in the world of $\mathcal{F}'_{\text{PAKE-3}}$:

Environment $\widetilde{\mathcal{Z}}$:

1. Initialize a single session between P and P' on pw. That is, pick any sid, and input $(\mathsf{NewSession}, sid, \mathsf{P}, \mathsf{P}', \mathsf{pw}, \mathsf{role})$ to P and $(\mathsf{NewSession}, sid, \mathsf{P}', \mathsf{P}, \mathsf{pw}, \mathsf{role}')$ to P'.
2. Instruct $\mathcal{A}$ to be an eavesdropper in session sid.
3. Output 1 if P outputs $(sid, K \in \{0,1\}^\lambda)$ and P' outputs $(sid, K' \in \{0,1\}^\lambda)$ before they halt, and output 0 otherwise.

Let $\mathcal{S}$ be a successful simulator for $\widetilde{\mathcal{Z}}$. In the ideal world, note that the extra clause in $\mathcal{F}'_{\text{PAKE-3}}$ (under the $\mathsf{NewSession}$ command) guarantees that $(\mathsf{NewKey}, sid, \mathsf{P}, \cdot)$ and $(\mathsf{NewKey}, sid, \mathsf{P}', \cdot)$ will be called at least once, causing P and P' to output a key in $\{0,1\}^\lambda$ (together with sid). Thus, $\widetilde{\mathcal{Z}}$ always outputs 1. This means that in the real world,

$$\Pr[\mathsf{P} \text{ outputs } (sid, K \in \{0,1\}^\lambda) \wedge \mathsf{P}' \text{ outputs } (sid, K' \in \{0,1\}^\lambda)] \geq 1 - \mathbf{Dist}(\mathcal{S}, \mathcal{Z}),$$

which is overwhelming. This shows that $\widetilde{\Pi}$ has guaranteed output.

$\mathcal{F}_{PAKE\text{-}3}$ with guaranteed output $\Rightarrow \mathcal{F}'_{PAKE\text{-}3}$: Given a successful simulator $\widetilde{\mathcal{S}}$ for $\widetilde{\Pi}$ realizing $\mathcal{F}_{\text{PAKE-3}}$ using $\mathcal{F}_{\text{AUTH}}$, we define a simulator $\widetilde{\mathcal{S}}'$ for $\widetilde{\Pi}$ realizing $\mathcal{F}'_{\text{PAKE-3}}$ using $\mathcal{F}'_{\text{AUTH}}$:

Simulator $\widetilde{\mathcal{S}}'$:

1. Run $\widetilde{\mathcal{S}}$, and processing each message as if $\widetilde{\mathcal{S}}'$ had received it.
2. Whenever $\widetilde{\mathcal{S}}$ sends $(sid, \mathsf{S}, \mathsf{R}, m)$ to $\mathcal{A}$, instruct $\widetilde{\mathcal{S}}$ to deliver (sid, S, m) to the recipient R.

Let $\widetilde{\mathcal{Z}}'$ be an efficient environment against $\widetilde{\Pi}$ realizing $\mathcal{F}'_{\text{PAKE-3}}$. Consider the following environment $\widetilde{\mathcal{Z}}$ against $\widetilde{\Pi}$ realizing $\mathcal{F}'_{\text{PAKE-3}}$:

Environment $\widetilde{\mathcal{Z}}$:

1. Run $\widetilde{\mathcal{Z}}'$, and pass messages between $\widetilde{\mathcal{Z}}'$ and P, P' and $\mathcal{F}'_{\text{AUTH}}$ without any modifications (when a message is sent from $\mathcal{F}_{\text{AUTH}}$, pass this message to $\widetilde{\mathcal{Z}}'$ as if it is from $\mathcal{F}'_{\text{AUTH}}$).
2. Whenever a message $(sid, \mathsf{S}, \mathsf{R}, m)$ is sent from $\mathcal{F}_{\text{AUTH}}$ to $\widetilde{\mathcal{Z}}'$, wait until $\widetilde{\mathcal{Z}}'$ finishes processing the message and all actions it triggers complete. Then let $\mathcal{F}_{\text{AUTH}}$ deliver the message to its destination R.
3. When $\widetilde{\mathcal{Z}}'$ outputs a bit b, output the same bit b.

In the real world, $\widetilde{\mathcal{Z}}$ behaves identically to $\widetilde{\mathcal{Z}}'$. Indeed, waiting for $\widetilde{\mathcal{Z}}'$'s processing (and whatever $\widetilde{\mathcal{Z}}'$ triggers) to finish before delivering the message is exactly the semantics given above for the execution splitting in $\mathcal{F}'_{\text{AUTH}}$ (see discussion at the start of this section).

In the ideal world, this same feature of $\widetilde{\mathcal{Z}}$ matches with $\widetilde{\mathcal{S}}'$ always instructing $\widetilde{\mathcal{S}}$ to simulate delivering its messages. This makes the ideal world with $\mathcal{F}'_{\text{PAKE-3}}$, $\widetilde{\mathcal{S}}'$ and $\widetilde{\mathcal{Z}}'$, and the ideal world with $\mathcal{F}_{\text{PAKE-3}}$, $\widetilde{\mathcal{S}}$ and $\widetilde{\mathcal{Z}}$, differ only in the extra clause in $\mathcal{F}'_{\text{PAKE-3}}$. However, this clause only matters when all three parties are honest, and no NewKey message has been sent for both P and P' — in which case in the ideal world with $\mathcal{F}_{\text{PAKE-3}}$, $\widetilde{\mathcal{S}}$ and $\widetilde{\mathcal{Z}}$, $\widetilde{\mathcal{Z}}$ might halt without the key sent to P and P'. We have that

$$
\begin{aligned}
&\mathbf{Dist}_{\widetilde{\Pi},\mathcal{F}'_{\text{PAKE-3}}}(\widetilde{\mathcal{S}}', \widetilde{\mathcal{Z}}') \\
&= \Pr[\mathsf{P}, \mathsf{P}' \text{ outputs nothing in some session in the ideal world with } \mathcal{F}_{\text{PAKE-3}}, \widetilde{\mathcal{S}} \text{ and } \widetilde{\mathcal{Z}}] \\
&\leq \Pr[\mathsf{P}, \mathsf{P}' \text{ outputs nothing in some session in the real world with } \widetilde{\Pi} \text{ and } \widetilde{\mathcal{Z}}] + \\
&\quad \mathbf{Dist}_{\widetilde{\Pi},\mathcal{F}_{\text{PAKE-3}}}(\widetilde{\mathcal{S}}, \widetilde{\mathcal{Z}}),
\end{aligned}
$$

which is negligible since $\widetilde{\Pi}$ has guaranteed output and $\widetilde{\mathcal{S}}$ is successful. So $\widetilde{\mathcal{S}}'$ is successful and we conclude that $\widetilde{\Pi}$ realizes $\mathcal{F}'_{\text{PAKE-3}}$. ■

7 Conclusion

In this work, we presented a comprehensive study of correctness in universally composable symmetric PAKE. Our contributions are four-fold:

First, we showed that TrivialPAKE, a protocol where the two parties simply output independent random keys, realizes the standard UC PAKE functionality $\mathcal{F}_{\text{PAKE}}$. The crux of the proof is that the simulator can use the $\mathsf{TestPwd}$ command in $\mathcal{F}_{\text{PAKE}}$ to interrupt protocol sessions, causing $\mathcal{F}_{\text{PAKE}}$ to output independent random keys to the two parties.

Second, we showed nine possible ways to address the issue. The first one is to add a separate notion of correctness. The others require the UC simulator to be *reasonable*: roughly speaking, the simulator is not allowed to send the $\mathsf{TestPwd}$ command to $\mathcal{F}_{\text{PAKE}}$ if the real adversary is an eavesdropper. There are eight ways to impose this constraint on the simulator, resulting from three dimensions:

– Whether it is a *strong* reasonable simulator, which means that it cannot send TestPwd before the real adversary tampers with a protocol message, even if the adversary later turns out not to be an eavesdropper;
– Whether it is a *perfectly* reasonable simulator, which means that it must have zero probability of sending TestPwd when it should not, rather than merely negligible probability;
– Whether we require only one successful simulator to be reasonable, or all successful simulators to be reasonable.

We proved that six of the approaches above are equivalent to directly enforcing correctness, while the other two are unachievable.

Third, we proved that it is impossibly to modify the UC PAKE functionality to include correctness. The high-level idea is that any UC PAKE functionality must have some mechanism equivalent to TestPwd in $\mathcal{F}_{\text{PAKE}}$, hence allowing TrivialPAKE's simulator to cause the two parties to output independent random keys.

Finally, we showed how to bypass the impossibility result by modeling PAKE as a three-party protocol, including a third party called the router. Messages between the two protocol parties must pass through the router via a pair of authenticated channels — hence a corrupted router essentially models the man-in-the-middle adversary. We presented a three-party PAKE functionality where the TestPwd command may only be sent from a corrupted party, who must be one of the three participants (including the router), instead of the ideal adversary. We proved that this PAKE functionality is equivalent to normal PAKE with correctness.

While this work is in the context of PAKE, it seems that similar issues about correctness appear in *any* protocol in the man-in-the-middle setting. For example, the UC authenticated key exchange (AKE) functionalities in [17, 25] also do not guarantee correctness: the functionalities include an Interfere command which causes the corresponding party to output a random key, just like TestPwd in $\mathcal{F}_{\text{PAKE}}$ (when the password guess is incorrect); thus, an incorrect protocol where the two parties output independent random keys still has a successful simulator, just as with TrivialPAKE. We conjecture that results similar to ours — including the natural fix by switching to the router model — hold for other UC functionalities in the man-in-the-middle setting.

Finally, as pointed out in Remark 2, reasonable simulators have been (informally) discussed while addressing another definitional issue in *asymmetric* PAKE (aPAKE), although not in the context of correctness (hence the underlying reason why this constraint on the UC simulator is needed is different from ours). It would be interesting to explore whether a similar impossibility result holds for aPAKE, and whether the requirement that a simulator be reasonable can be removed by adding another party to the modeling of aPAKE.

References

1. M. Abdalla, M. Barbosa, T. Bradley, S. Jarecki, J. Katz, and J. Xu. Universally composable relaxed password authenticated key exchange. In *CRYPTO 2020, Part I*, Aug. 2020.
2. M. Abdalla, B. Haase, and J. Hesse. Security analysis of CPace. In *ASIACRYPT 2021, Part IV*, Dec. 2021.
3. M. Abdalla and D. Pointcheval. Simple password-based encrypted key exchange protocols. In *CT-RSA 2005*, Feb. 2005.
4. M. Bellare, D. Pointcheval, and P. Rogaway. Authenticated key exchange secure against dictionary attacks. In *EUROCRYPT 2000*, May 2000.
5. S. M. Bellovin and M. Merritt. Encrypted key exchange: Password-based protocols secure against dictionary attacks. In *1992 IEEE Symposium on Security and Privacy*, May 1992.
6. S. M. Bellovin and M. Merritt. Augmented encrypted key exchange: A password-based protocol secure against dictionary attacks and password file compromise. In *ACM CCS 93*, Nov. 1993.
7. V. Boyko, P. D. MacKenzie, and S. Patel. Provably secure password-authenticated key exchange using Diffie-Hellman. In *EUROCRYPT 2000*, May 2000.
8. R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd FOCS*, Oct. 2001.
9. R. Canetti and M. Fischlin. Universally composable commitments. In *CRYPTO 2001*, Aug. 2001.
10. R. Canetti, S. Halevi, J. Katz, Y. Lindell, and P. D. MacKenzie. Universally composable password-based key exchange. In *EUROCRYPT 2005*, May 2005.
11. R. Canetti, E. Kushilevitz, and Y. Lindell. On the limitations of universally composable two-party computation without set-up assumptions. In *EUROCRYPT 2003*, May 2003.
12. R. Canetti, Y. Lindell, R. Ostrovsky, and A. Sahai. Universally composable two-party and multi-party secure computation. In *34th ACM STOC*, May 2002.
13. Crypto Forum Research Group. PAKE selection, 2020. `https://github.com/cfrg/pake-selection`.
14. P.-A. Dupont, J. Hesse, D. Pointcheval, L. Reyzin, and S. Yakoubov. Fuzzy password-authenticated key exchange. In *EUROCRYPT 2018, Part III*, Apr. / May 2018.
15. C. Gentry, P. MacKenzie, and Z. Ramzan. A method for making password-based key exchange resilient to server compromise. In *CRYPTO 2006*, Aug. 2006.
16. A. Groce and J. Katz. A new framework for efficient password-based authenticated key exchange. In *ACM CCS 2010*, Oct. 2010.
17. Y. Gu, S. Jarecki, and H. Krawczyk. KHAPE: Asymmetric PAKE from key-hiding key exchange. In *CRYPTO 2021, Part IV*, Aug. 2021.
18. B. Haase and B. Labrique. AuCPace: Efficient verifier-based PAKE protocol tailored for the IIoT. Cryptology ePrint Archive, Report 2018/286, 2018. `https://eprint.iacr.org/2018/286`.
19. J. Hesse. Separating symmetric and asymmetric password-authenticated key exchange. In *SCN 20*, Sept. 2020.
20. S. Jarecki, H. Krawczyk, and J. Xu. OPAQUE: An asymmetric PAKE protocol secure against pre-computation attacks. In *EUROCRYPT 2018, Part III*, Apr. / May 2018.

21. J. Katz, U. Maurer, B. Tackmann, and V. Zikas. Universally composable synchronous computation. In *TCC 2013*, Mar. 2013.
22. J. Katz, R. Ostrovsky, and M. Yung. Efficient password-authenticated key exchange using human-memorable passwords. In *EUROCRYPT 2001*, May 2001.
23. J. Katz and V. Vaikuntanathan. Smooth projective hashing and password-based authenticated key exchange from lattices. In *ASIACRYPT 2009*, Dec. 2009.
24. I. McQuoid, M. Rosulek, and L. Roy. Minimal symmetric PAKE and 1-out-of-N OT from programmable-once public functions. In *ACM CCS 2020*, Nov. 2020.
25. B. F. D. Santos, Y. Gu, S. Jarecki, and H. Krawczyk. Asymmetric PAKE with low computation and communication. In *EUROCRYPT 2022, Part II*, May / June 2022.
26. S. V. Smyshlyaev. Results of the PAKE selection process, 2020. `https://mailarchive.ietf.org/arch/msg/cfrg/LKbwodpa5yXo6VuNDU66vt_Aca8`.

A Proof of (1)⇒(3) in Lemma 1

Proof. Let $\mathcal{S}$ be any successful simulator for Π and $\mathcal{Z}$ be any efficient environment; it is not hard to see that $\Pr[\mathsf{NoOutput}(\mathcal{S},\mathcal{Z})] \leq \Pr[\overline{\mathsf{Correct}(\mathsf{pw})}] + \mathbf{Dist}(\mathcal{S},\mathcal{Z})$ is negligible, and we need to show that $\Pr[\mathsf{SpuriousGuess}(\mathcal{S},\mathcal{Z})]$ is negligible. Let q be an upper bound of the total number of sessions that $\mathcal{Z}$ initializes. We construct another environment $\mathcal{Z}'$, which behaves exactly as $\mathcal{Z}$ except that $\mathcal{Z}'$ uses a random password in a random session:

Environment $\mathcal{Z}'$:

1. Sample a random integer $\ell \leftarrow [q]$ as a guess of the session in which $\Pr[\mathsf{SpuriousGuess}(\mathcal{S},\mathcal{Z})]$ occurs.
2. Activate $\mathcal{Z}$. For all sessions other than the ℓ-th, pass messages between $\mathcal{Z}$ and P, as well as messages between $\mathcal{Z}$ and P', without any modifications.
3. For the ℓ-th session sid, sample a random password $\mathsf{pw} \in \mathsf{Dict}$ and input $(\mathsf{NewSession}, sid, \mathsf{P}, \mathsf{P}', \mathsf{pw}, \mathsf{role})$ to P and $(\mathsf{NewSession}, sid, \mathsf{P}', \mathsf{P}, \mathsf{pw}, \mathsf{role}')$ to P'. After that, instruct $\mathcal{A}$ to be an eavesdropper.
4. When P outputs (sid, K) and P' outputs (sid, K') in the ℓ-th session, output 1 if $K = K' \in \{0,1\}^\lambda$ and output 0 otherwise. If P or P' does not output anything when it halts, then output 0.

Note that the output of $\mathcal{Z}'$ only depends on the ℓ-th session, in which the password is pw and $\mathcal{Z}'$ instructs the adversary to be an eavesdropper. In the real world, we have that

$$\Pr[\mathcal{Z} \text{ outputs 1 in the real world}] = \Pr[\mathsf{Correct}(\mathsf{pw})].$$

In the ideal world, assume that the guess of $\mathcal{Z}'$ is correct, i.e., $\mathsf{SpuriousGuess}(\mathcal{S},\mathcal{Z})$ occurs in the ℓ-th session (denote this event as $\mathsf{SpuriousGuess}_\ell(\mathcal{S},\mathcal{Z})$). This means that that $\mathcal{Z}$ instructs the adversary to be an eavesdropper in this session, and $\mathcal{S}$ sends $(\mathsf{TestPwd}, sid, \mathsf{P}, \mathsf{pw}^*)$ or

$(\mathsf{TestPwd}, sid, \mathsf{P}', \mathsf{pw}^*)$ to $\mathcal{F}_{\mathrm{PAKE}}$. Observe that until the end of the ℓ-th session, the view of $\mathcal{S}$ in the experiment simulated by $\mathcal{Z}'$ is identical to the view of $\mathcal{S}$ in the experiment with $\mathcal{Z}$ (the only difference between the behaviors of $\mathcal{Z}'$ and $\mathcal{Z}$ is that $\mathcal{Z}'$ uses pw as the password in the ℓ-th session, in which both $\mathcal{Z}$ and $\mathcal{Z}'$ instruct the adversary as an eavesdropper, so the password in this session is independent of the view of $\mathcal{S}$); thus, $\mathcal{S}$ also sends $(\mathsf{TestPwd}, sid, \mathsf{P}, \mathsf{pw}^*)$ or $(\mathsf{TestPwd}, sid, \mathsf{P}', \mathsf{pw}^*)$ to $\mathcal{F}_{\mathrm{PAKE}}$ in the experiment simulated by $\mathcal{Z}'$. Since pw is independent of $\mathcal{S}$'s view, the probability that $\mathsf{pw}^* = \mathsf{pw}$ is $1/|\mathsf{Dict}|$, and if $\mathsf{pw}^* \neq \mathsf{pw}$ (i.e., the password guess of $\mathcal{S}$ is wrong), then the corresponding session record becomes $\mathsf{interrupted}$ and the party outputs a random key in $\{0,1\}^\lambda$, as can be seen in the third case of NewKey — so the probability that $K = K'$ is $1/2^\lambda$. Summing up, we have that

$$\Pr[K = K' \text{ in the ideal world} \mid \mathsf{SpuriousGuess}_\ell(\mathcal{S}, \mathcal{Z})] \leq \frac{1}{|\mathsf{Dict}|} + \frac{1}{2^\lambda},$$

so

$$\begin{aligned}
&\Pr[\mathcal{Z} \text{ outputs 1 in the ideal world}] \\
\leq &\Pr[K = K' \text{ in the ideal world}] \\
\leq &\Pr[K = K' \text{ in the ideal world} \mid \mathsf{SpuriousGuess}_\ell(\mathcal{S}, \mathcal{Z})] \cdot \Pr[\mathsf{SpuriousGuess}_\ell(\mathcal{S}, \mathcal{Z})] \\
&\quad + \Pr[\overline{\mathsf{SpuriousGuess}_\ell(\mathcal{S}, \mathcal{Z})}] \\
= &\left(\frac{1}{|\mathsf{Dict}|} + \frac{1}{2^\lambda}\right) \cdot \Pr[\mathsf{SpuriousGuess}_\ell(\mathcal{S}, \mathcal{Z})] + (1 - \Pr[\mathsf{SpuriousGuess}_\ell(\mathcal{S}, \mathcal{Z})]) \\
= &1 - \left(1 - \frac{1}{|\mathsf{Dict}|} - \frac{1}{2^\lambda}\right) \cdot \Pr[\mathsf{SpuriousGuess}_\ell(\mathcal{S}, \mathcal{Z})] \\
= &1 - \left(1 - \frac{1}{|\mathsf{Dict}|} - \frac{1}{2^\lambda}\right) \cdot \frac{\Pr[\mathsf{SpuriousGuess}(\mathcal{S}, \mathcal{Z})]}{q}.
\end{aligned}$$

Comparing the probabilities that $\mathcal{Z}$ outputs 1 in the real world and in the ideal world, we get

$$\mathbf{Dist}(\mathcal{S}, \mathcal{Z}) \geq \left(1 - \frac{1}{|\mathsf{Dict}|} - \frac{1}{2^\lambda}\right) \cdot \frac{\Pr[\mathsf{SpuriousGuess}(\mathcal{S}, \mathcal{Z})]}{q} - \Pr[\overline{\mathsf{Correct}(\mathsf{pw})}],$$

so

$$\Pr[\mathsf{SpuriousGuess}(\mathcal{S}, \mathcal{Z})] \leq \frac{q}{1 - \frac{1}{|\mathsf{Dict}|} - \frac{1}{2^\lambda}} \cdot (\mathbf{Dist}(\mathcal{S}, \mathcal{Z}) + \Pr[\overline{\mathsf{Correct}(\mathsf{pw})}]).$$

Since Π is correct, $\Pr[\overline{\mathsf{Correct}(\mathsf{pw})}]$ is negligible; since $\mathcal{S}$ is successful, $\mathbf{Dist}(\mathcal{S}, \mathcal{Z})$ is negligible. Therefore, we conclude that $\Pr[\mathsf{SpuriousGuess}(\mathcal{S}, \mathcal{Z})]$ is negligible, completing the proof. ■