Does the UC-Security Notion for PAKE Imply Game-Based Security? *

Jiayu Xu

Oregon State University, xujiay@oregonstate.edu

Abstract

A Password-Authenticated Key Exchange (PAKE) protocol allows two parties to jointly establish a cryptographically strong key, in the setting where the only information shared in advance is a low-entropy "password". The two standard security definitions for PAKE are the game-based one by Bellare, Pointcheval and Rogaway (BPR-security, EUROCRYPT 2000) and the Universally Composable (UC) one by Canetti et al. (EUROCRYPT 2005). It is well-known that UC-security implies BPR-security; however, there are a large number of variants of both definitions, and the relation between them is not entirely clear.

In this work, we thoroughly study a variant of BPR-security by Katz, Ostrovsky and Yung (KOY-security, JACM 2009):

1. We show, via a counterexample, that UC-security does *not* imply KOY-security;
2. We then prove that a variant of UC-security, called implicit-only UC-security (Dupont et al., EUROCRYPT 2018), implies KOY-security.

Interestingly, we make the observation that KOY- and implicit-only UC-security essentially strengthen their standard counterparts in the same manner. We also present detailed explanations of all four security notions.

1 Introduction

A *Password-Authenticated Key Exchange (PAKE)* protocol allows two parties to jointly establish a cryptographically strong key, in the setting where the only information shared in advance is a low-entropy string called the password. Crucially, such a protocol must be secure against man-in-the-middle adversaries that can arbitrarily modify protocol messages. One common attack on PAKE is *online guessing*, in which the adversary chooses a candidate password pw^* and impersonates Bob to Alice; if Alice's password is indeed pw^*, then the adversary

*The `.tex` code of this paper can be found at https://github.com/jiayux123/game-based-pake.

learns Alice's output key. Since passwords have low entropy, the adversary has a non-negligible probability of guessing Alice's password correctly; at a high level, security for PAKE requires that online guessing is the only feasible attack.

While the intuition of PAKE security looks simple and innocuous, a formal security definition proves incredibly difficult. There are two such definitions that are considered standard: the game-based security notion by Bellare, Pointcheval and Rogaway [BPR00] (henceforth *BPR-security*), and the Universally Composable (UC) security notion by Canetti et al. [CHK+05] (henceforth *UC-security*). Both definitions take pages to describe and explain, and contain various subtleties that are confusing to beginners and understudied by the community. A standard result [CHK+05, Appendix A] shows that UC-security implies BPR-security.

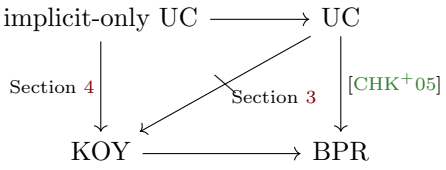
Over the years, a large number of variants of both BPR- and UC-security have been proposed. While many of these are essentially tailoring the definition to fit a specific protocol construction, some do have an intuitive interpretation and capture certain nuances of PAKE security. This work focuses on two of them:

- The variant of BPR-security by Katz, Ostrovsky and Yung [KOY09] (henceforth *KOY-security*), which places more constraints on what exactly counts as an "online attack"; and
- The variant of UC-security by Dupont et al. [DHP+18] called *implicit-only UC-security*, which disallows the ideal adversary from learning whether an online attack has succeeded or not.

Neither of the aforementioned works' main focus was on security notions. The conference proceedings version of the KOY paper [KOY01] proposed a new PAKE protocol and proved its BPR-security; in the journal version [KOY09] KOY proved that their protocol actually satisfies the stronger KOY-security, although the difference between these two notions was left unexplained. (The KOY-security notion has since been used in some subsequent works [KV11].) Implicit-only UC-security was introduced because the authors of [DHP+18] used PAKE to construct what's called fuzzy PAKE, and the underlying PAKE needs to satisfy this stronger security notion for their fuzzy PAKE protocols to be secure. Perhaps this explains why these two security notions are not widely used: we do not know of any studies of either KOY- or implicit-only UC-security per se, and subsequent works generally continued to use BPR- and UC-security.

Our contributions. In this work, we present a thorough study of KOY- and implicit-only UC-security for PAKE, and draw some interesting connections between them and their more standard counterparts. Our technical contributions are two-fold:

1. In Section 3 we show that UC-security does *not* imply KOY-security, by giving a concrete counterexample that is UC-secure but not KOY-secure;
2. In Section 4 we show that implicit-only UC-security implies KOY-security.



Finally, in Section 5 we point out that KOY- and implicit-only UC-security essentially strengthen their standard counterparts in the same manner, namely [click to reveal spoiler]

. In this way, we give an interpretation of what exactly the KOY- and implicit-only UC-security notions entail.

As a side contribution, in Section 2 we discuss the four security definitions in detail — not only *what* exactly these definitions are, but also the *rationales* behind certain design choices in the definitions. By doing this, we hope to help "demystify" PAKE security notions and make them more friendly to beginners; as such, this work also serves as a survey (or tutorial) of different PAKE security definitions — which is incomplete but does cover the standard ones that are most commonly used.

A personal anecdote: how did I dig out KOY-security? I learned game-based PAKE security definitions "backwards". For a very long time, KOY-security was the standard reference for me, and I had no idea that KOY and BPR were actually different. Very recently, I needed to prove a generalization of UC-security $\Rightarrow$ game-based security, and I just couldn't make the proof go through. After some thought, I realized that the "theorem" I tried to prove was actually wrong — similar to the counterexample in Section 3 — and you needed to tweak the game-based security definition to make the proof work.

At that time I thought there was an error in the standard result in [CHK+05]. [CHK+05] claims that the game-based security notion they use is BPR, so I checked [BPR00]. I immediately realized that the "BPR-security" I had in mind was in fact not what BPR said, and the actual BPR-security was exactly what I needed in my proof.

I decided to write down my counterexample. During this process, I realized that apart from replacing KOY- with BPR-security, another way to invalidate the counterexample would be to strengthen UC-security to implicit-only UC-security. Perhaps implicit-only UC-security implies KOY-security? This proof indeed went through, and the technical pieces of this work were complete.

2 Preliminaries

Let λ be the security parameter. Most notations in this work are standard: for example, $x \leftarrow S$ means sampling an element x uniformly at random from set S; $x := a$ means assigning value a to variable x; PPT means "probabilistic

polynomial-time"; and negl denotes a negligible function in λ. We assume the reader is familiar with the UC framework [Can01].

2.1 Overview of Security Definitions for PAKE

We first give an informal, high-level overview of the ideas behind the security definitions for PAKE; what's said in this section applies to both the game-based and UC definitions. A few paragraphs in this section and Section 2.3 are taken almost verbatim from [Xu25].

Any reasonable security definition for PAKE must formalize the following two principles:

> *The first principle: The adversary can only perform a single online guessing attack per protocol instance.*
>
> *The second principle: All output keys are independent of each other, except in cases where correlation is inevitable.*

Correspondingly, the definition must contain the following parts (which is where the technical complications come from):

1. What exactly constitutes an "online guessing attack" must be defined.
2. Each protocol instance should output a uniformly random key independent of everything else, except that the cases where correlation is inevitable need to be singled out and handled separately.

The second principle warrants further explanation. What are those cases where correlation of output keys is inevitable? There are two of them:

- If the adversary succeeds in an online attack, then of course it can learn the attacked instance's output key. In this case all security guarantees for this instance are lost and we cannot claim independence;
- If the adversary is passive between two instances (i.e., it merely forwards all protocol messages without any modification), then of course these two instances output the same key. (But this key should still be independent of other instances' output keys.)[1]

We will see that the above forms the core of both the game-based and UC definitions; however, they formalize these two principles in drastically different ways.

[1] This assumes that the two instances' passwords match. As we will see, in the game-based definitions we always assume this, whereas in the UC definitions we additionally consider the case where the two instances (that intend to connect to each other) hold different passwords.

2.2 BPR- And KOY-Security Definitions for PAKE

Setup. The system consists of a number of protocol parties P, and each party can have a number of instances P^i. An instance can send protocol messages to an instance that belongs to another party.

Let D be the set of all candidate passwords (the "dictionary"); we assume $|D| \geq 2$ and is polynomial, and D is public. For each pair of parties (P, P'), sample $\mathsf{pw} \leftarrow D$ as the shared password between them.[2] (Note that we do not consider the case where the two instances hold different passwords.)

Protocol execution. The security notion for PAKE is formalized as a game between a PPT adversary $\mathcal{A}$ and a game challenger, where the challenger runs the protocol parties (and their instances) and sends to $\mathcal{A}$ certain information that $\mathcal{A}$ is supposed to learn during an attack. Concretely, $\mathcal{A}$ can send the following commands to the game challenger:

- $\mathsf{Execute}(P^i, (P')^j)$ — This models an eavesdropping attack, where the adversary is passive and learns all protocol messages sent between the two instances (the "protocol transcript"). Concretely, if neither P^i nor $(P')^j$ is used, the game challenger runs the protocol between P^i and $(P')^j$, and returns the transcript to $\mathcal{A}$. Once a $\mathsf{Execute}(P^i, (P')^j)$ command is sent, the game challenger marks both P^i and $(P')^j$ as used.

- $\mathsf{Send}(P^i, (P')^j, m^*)$ — This models a man-in-the-middle attack, where the adversary sends message m^* to $(P')^j$, as if the message is from P^i (below we simply say "the adversary sends message m^* from P^i to $(P')^j$"). Concretely, the game challenger sends message m^* from P^i to $(P')^j$; if $(P')^j$ sends a protocol message m' then the game challenger returns m' to $\mathcal{A}$, otherwise (i.e., m^* is the last message in the protocol causing $(P')^j$ to terminate) the game challenger returns $\perp$ to $\mathcal{A}$.
  As a special case, $\mathsf{Send}(P^i, (P')^j, \text{start})$ models the adversary initiating instance P^i and sees its first message (intended for $(P')^j$). Concretely, if P^i is not used, the game challenger initiates instance P^i and returns P^i's message to $\mathcal{A}$. Once a $\mathsf{Send}(P^i, (P')^j, \text{start})$ command is sent, the game challenger marks P^i as used.[3]

Note that how we handle a used instance (i.e., an instance that has been initiated) guarantees that an instance cannot be run twice; in particular, $\mathcal{A}$ cannot

[2] We require $P \neq P'$, i.e., only instances that belong to different parties may communicate. Both [BPR00] and [KOY09] partition all parties into two categories, clients and servers, and a client instance must communicate with a server instance (and vice versa). Our treatment is slightly more general (e.g., it allows Alice to be a client while talking to Bob and a server while talking to Carol).

[3] Our specification assumes that if $(P')^j$ is the responder, i.e., $(P')^j$ waits for P^i's message before sending its own message, then the first of $\mathsf{Send}((P')^j, P^i, \text{start})$ and $\mathsf{Send}(P^i, (P')^j, m^*)$ returns $\perp$ and the second returns $(P')^j$'s message to P^i. (In other words, in order to see the responder's message, $\mathcal{A}$ has to send both a "start" command for the responder and a message from the initiator to the responder.) This slightly deviates from [BPR00, KOY09] but fits the UC-security definition better.

send both Execute and Send on the same instance. Therefore, all used instances can be partitioned into two types: Execute instances and Send instances.

- $\mathsf{Reveal}(P^i)$ — This models the adversary learning a completed instance's output key, via e.g., a side-channel attack. Concretely, if P^i has computed its output key SK_i, the game challenger returns SK_i to $\mathcal{A}$; otherwise the game challenger returns $\perp$ to $\mathcal{A}$.

This completes the description of what $\mathcal{A}$ can see while attacking the protocol.

Security. We still need to decide whether $\mathcal{A}$ wins the game. Similar to the security definition for unauthenticated key exchange, we let $\mathcal{A}$ choose an instance and try to distinguish its output key with a random string:

- $\mathsf{Test}(P^i)$ — The game challenger samples a bit $b \leftarrow \{0,1\}$. If P^i has computed its output key SK_i and $b = 1$, the game challenger returns SK_i to $\mathcal{A}$; if P^i has computed its output key SK_i and $b = 0$, the game challenger returns a uniformly random string in $\{0,1\}^\lambda$ to $\mathcal{A}$; otherwise the game challenger returns $\perp$ to $\mathcal{A}$. The Test command can be sent only once.

At the end of the game, $\mathcal{A}$ outputs a bit b^* and wins if $b^* = b$.

If we stop here then there are a few ways for $\mathcal{A}$ to win trivially, which we must disallow:

1. If $\mathcal{A}$ sends $\mathsf{Reveal}(P^i)$ and $\mathsf{Test}(P^i)$ for the same P^i (no matter what the order is), then $\mathcal{A}$ knows that $b = 1$ if and only if they return the same result (except with negligible probability — same below).
2. If $\mathcal{A}$ sends $\mathsf{Execute}(P^i, (P')^j)$, then P^i and $(P')^j$'s output keys are the same. Then if $\mathcal{A}$ Reveals one of them and Tests the other, then again $b = 1$ if and only if they return the same result.
3. Note that Reveal is not the only way for $\mathcal{A}$ to be passive: if both P^i and $(P')^j$ are Send instances, and their views are identical (i.e., $\mathcal{A}$ forwards all messages between them via Send commands), then P^i and $(P')^j$'s output keys are again the same. If $\mathcal{A}$ Reveals one of them and Tests the other, then $b = 1$ if and only if they return the same result.[4]

We need to prohibit all these behaviors, i.e., we only quantify over all PPT adversaries that do not do any of 1–3 above.[5]

[4] This does not consider the case where $\mathcal{A}$ is "effectively passive" by merely modifying the *format* of protocol messages. For example, if a protocol message contains a bit bit that is ignored by the receiver, then $\mathcal{A}$ can flip bit while transmitting this message; this should be treated as if $\mathcal{A}$ is passive. We do not consider such degenerate cases, as they are inconsequential to our main results; for how to talk about such cases formally, see the discussion about "session ID" (not to be confused with the session ID in the UC framework) in [BPR00, KOY09].

[5] [BPR00, KOY09] define two instances to be *partnered* if their transcripts are the same. Using this term, 1–3 above can be succinctly stated as: For the instance that $\mathcal{A}$ Tests, $\mathcal{A}$ cannot Reveal either the instance itself or its partner, no matter whether Reveal happens first or Test happens first.

Note that 2 and 3 above exactly reflect the second principle in Section 2.1: if $\mathcal{A}$ is passive between P^i and $(P')^j$, then their output keys will be the same, and we need to place some restrictions on what $\mathcal{A}$ can Test. PAKE security should mean that except for these cases, the instances' output keys are uniformly random and independent of everything else, so $\mathcal{A}$ can Test any one of them and the real key should be indistinguishable from a random string — unless $\mathcal{A}$ succeeds in an online attack, which we define next.

The "baseline" winning probability. The last piece of the definition is to define $\mathcal{A}$'s "baseline" winning probability, i.e., $\mathcal{A}$'s winning probability if it only performs inevitable attacks. The first principle in Section 2.1 states that each instance should allow only one online attack, in which $\mathcal{A}$ can check one password guess (and gain $1/|D|$ advantage); therefore, if there are q instances that are online attacked, then the "baseline" winning probability should be $1/2+q/2|D|$.

But when exactly is an instance "online attacked"? This is where BPR- and KOY-security diverge. To begin with, if the instance is not even used (i.e., initiated) then of course there is no way to attack it. For an Execute instance, since $\mathcal{A}$ is passive there and merely sees the transcript, that should not count as an online attack either.[6]

This leaves Send instances. In BPR-security, all Send instances are considered online attacked; in KOY-security, a Send instance is online attacked only if $\mathcal{A}$ also Reveals or Tests it. (The reader should pause a bit here and think about what this difference actually means; it should become clear after seeing the counterexample in Section 3, and we reveal our understanding in Section 5.) With the concept of "online attacks" defined, we can finally state our security definition for PAKE.

Definition 1. *A PAKE protocol is* BPR-secure/KOY-secure *if for any PPT adversary $\mathcal{A}$ that does not perform any of the prohibited behaviors 1–3 above,*

$$\Pr[\mathcal{A} \textit{ wins}] \leq \frac{1}{2} + \frac{q}{2|D|} + \mathsf{negl},$$

where q is the upper bound of the number of instances that are online attacked. The definition of an "online attacked instance" is the red text above for BPR-security and the blue text above for KOY-security.

We stress that the security game for BPR- and KOY-security is exactly the same; the only difference lies in what counts as an "online attack", which in turn affects the "baseline" winning probability. KOY-security is stronger because it places more restrictions on "online attacks", causing q — hence the "baseline" winning probability — to be potentially lower (so it is easier for $\mathcal{A}$ to "break" the protocol).

[6] This implies an important security property for PAKE: a PAKE protocol should be resilient to "offline dictionary attacks", i.e., a protocol transcript that the adversary passively observes should not tell the adversary any information about the password. For example, $H(\mathsf{pw})$ (where H is an RO) cannot be used as a protocol message.

2.3 UC- And Implicit-Only UC-Security Definitions for PAKE

UC-security definitions are done via a functionality that defines the ideal adversary's (a.k.a. the simulator's) interface. See the UC- and implicit-only UC functionalities for PAKE in Figure 1.

- On input $(\mathsf{NewSession}, sid, P, P', \mathsf{pw})$ from P, send $(\mathsf{NewSession}, sid, P, P')$ to $\mathcal{A}^*$. Furthermore, if this is the first NewSession message for sid, or this is the second NewSession message for sid and there is a record $\langle P', P, \star\rangle$, then store $\langle P, P', \mathsf{pw}\rangle$ and mark it fresh.
- On $(\mathsf{TestPwd}, sid, P, \mathsf{pw}^*)$ from $\mathcal{A}^*$, if there is a record $\langle P, P', \mathsf{pw}\rangle$ marked fresh, then do: // pw is P's password
  - If $\mathsf{pw}^* = \mathsf{pw}$, then mark the record compromised and send "correct guess" to $\mathcal{A}^*$.
  - If $\mathsf{pw}^* \neq \mathsf{pw}$, then mark the record interrupted and send "wrong guess" to $\mathcal{A}^*$.
- On $(\mathsf{NewKey}, sid, P, SK^* \in \{0,1\}^\lambda)$ from $\mathcal{A}^*$, if there is a record $\langle P, P', \mathsf{pw}\rangle$, and this is the first NewKey message for sid and P, then output (sid, SK) to P, where SK is defined as follows:
  - If the record is compromised, then set $SK := SK^*$.
  - If the record is fresh, a key (sid, SK') has been output to P', at which time there was a record $\langle P', P, \mathsf{pw}\rangle$ marked fresh, then set $SK := SK'$. // if no attack in this session, and passwords match, then keys should also match
  - Otherwise sample $SK \leftarrow \{0,1\}^\lambda$.

  Finally, mark the record completed. // cannot send TestPwd after instance completes

Figure 1: UC PAKE functionality $\mathcal{F}_{\text{PAKE}}$ (with red text) and implicit-only UC PAKE functionality $\mathcal{F}_{\text{iPAKE}}$ (red text omitted)

In the UC functionalities, party P's instance is represented by the record $\langle P, P', \mathsf{pw}\rangle$, where P' is P's counterparty and pw is P's password. A NewSession command initiates an instance; each session has two instances, so we allow for a NewSession command that initiates the $\langle P, P', \mathsf{pw}\rangle$ instance and another NewSession command that initiates the $\langle P', P, \mathsf{pw}'\rangle$ instance. Note that P and P''s passwords might be different.

Online guessing attacks are modeled by the TestPwd command, in which the ideal adversary specifies an instance and a password guess. Crucially, a TestPwd command can be sent only when the instance is fresh, and once such a command is sent, the instance will become compromised or interrupted (de-

pending on whether the password guess is correct) and never return to fresh. (Think of fresh as a shorthand for "unattacked", compromised as "successfully attacked", and interrupted as "unsuccessfully attacked".) In this way, it is guaranteed that at most one TestPwd command can be sent per instance: this is how the UC functionality reflects the first principle in Section 2.1.

TestPwd is also where the UC- and implicit-only UC-security definitions differ: in UC the ideal adversary knows whether its password guess is correct or not, whereas in implicit-only UC it doesn't. We will discuss what this entails in Section 5. Clearly, implicit-only UC-security is stronger than UC-security, as the simulator has less power in the former (making the simulation potentially harder).

Finally, a NewKey command allows an instance to complete and output its key. How this output key is generated exactly reflects the second principle in Section 2.1:

- If the instance is compromised, i.e., the adversary has succeeded in an online attack, then all security guarantees for this instance are lost. In this case, we simply let the ideal adversary set the output key (the first bullet).
- If both the current instance and its counterparty's instance are fresh, i.e., the adversary is passive between two instances, and their passwords match, then these two instances should output the same random key. Formalizing this is a little cumbersome: if P' outputs before P does, then P' should output a random key independent of everything else (a subcase of the third bullet), and when P outputs later, its key should be equal to P''s key (the second bullet).
- In all other cases, the instance's output key should be independent of everything else (the third bullet).

command	NewSession	$\longrightarrow$	TestPwd	$\longrightarrow$	NewKey
state	fresh	$\longrightarrow$	compromised or interrupted	$\longrightarrow$	completed

Figure 2: "Life cycle" of an instance. TestPwd can be skipped

Comparison with game-based definitions. Both game-based and UC-security definitions formalize the two principles in Section 2.1. However, they do so in very different ways.

The game-based definitions reflect the first principle in the "baseline" winning probability: an online attack should only increase the adversary's advantage by up to $1/|D|$. In contrast, the UC definition directly lets the ideal adversary specify a password guess while doing an online attack, and the ideal

adversary can send at most one TestPwd command per instance (subsequent such commands will be ignored).

For the second principle, the technical challenge is to rule out the special cases where output key correlation is inevitable. The game-based definitions prohibit the adversary from Revealing one instance and Testing its counterparty's instance, if the adversary is passive between the two instances; as for successfully attacked instances, the game simply "gives up" and allows the adversary to win (because this possibility is already covered by the "baseline" winning probability). The UC definitions again reflect this principle explicitly: the "successfully attacked instances" are handled in the first bullet under NewKey, and the "passive adversary" is handled in the second bullet.

Finally, the game-based definitions always assume two instances that connect to each other have the same password, whereas the UC definitions allow the two parties to have different passwords. In particular, what will happen if the adversary is passive between two instances with different passwords? The game-based definitions do not consider this situation, whereas the UC definitions specify that the two instances' output keys must be independently random in this case.

2.4 The EKE Protocol

The *Encrypted Key Exchange (EKE) protocol* [BM92] is the first PAKE protocol ever proposed. Here we will use EKE based on Diffie–Hellman, in which party P samples $x \leftarrow \mathbb{Z}_p$ and sends $c := \mathcal{E}(\mathsf{pw}, g^x)$ to its counterparty P', and on c' from P', P computes $Y := \mathcal{D}(\mathsf{pw}, c')$ and outputs $\mathrm{Hash}(Y^x)$ as the key; P' does the same. Here, $(\mathbb{G}, g, p)$ is a cyclic group with generator g and order p, $(\mathcal{E}, \mathcal{D})$ is an ideal cipher (where $\mathcal{E}$ is the IC encryption algorithm and $\mathcal{D}$ is the IC decryption algorithm), and Hash is an RO; both have range $\{0,1\}^\lambda$.

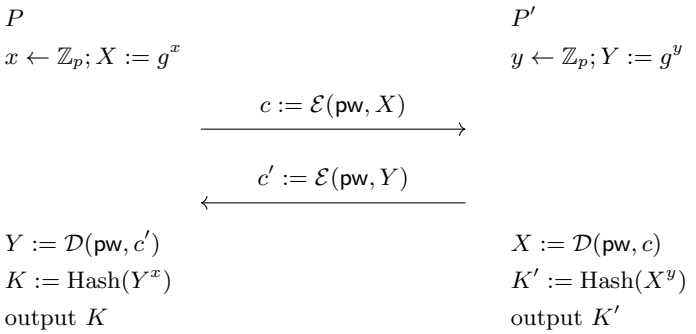


Figure 3: The EKE protocol

EKE has been proven to UC-realize the (standard) UC PAKE functionality $\mathcal{F}_{\mathrm{PAKE}}$ under the CDH assumption in $(\mathbb{G}, g, p)$ [JRX25]. We specify the EKE

protocol for concreteness only; it is only used in our counterexample in Section 3, and can be replaced by any "normal" PAKE protocol that UC-realizes $\mathcal{F}_{\text{PAKE}}$.

3 UC-Security Does Not Imply KOY-Security

3.1 The Counterexample

Consider the following PAKE protocol:

Protocol Π:

1. Protocol parties P and P' run EKE, resulting in keys K (for P) and K' (for P').
2. P sends $h := H_0(K)$ to P'.
3. P and P' output $H(K)$ and $H(K')$, respectively. Here, H and H_0 are two ROs from $\{0,1\}^\lambda$ to itself.

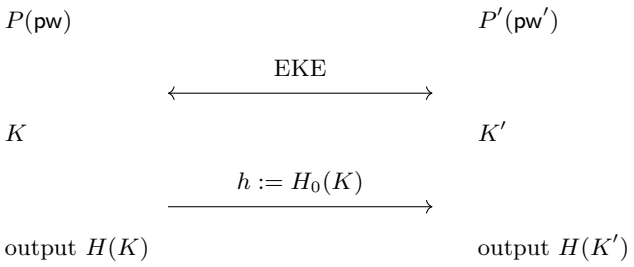


Figure 4: Our counterexample

Note that h is a redundant message, in the sense that neither party's output depends on it.

3.2 The Protocol Is UC-Secure

Proof. Since EKE UC-realizes $\mathcal{F}_{\text{PAKE}}$, we may replace the EKE part in protocol Π with $\mathcal{F}_{\text{PAKE}}$. Call the resulting protocol $\tilde{\Pi}$; we only need to prove that $\tilde{\Pi}$ is UC-secure. In order to distinguish the $\mathcal{F}_{\text{PAKE}}$ that is a building block of the *real protocol* $\tilde{\Pi}$ and the $\mathcal{F}_{\text{PAKE}}$ that is the functionality interacting with the simulator in the *ideal world*, we abbreviate the latter as $\mathcal{F}$.

Consider the following simulator $\mathcal{S}$. Roughly speaking, $\mathcal{S}$ simulates $\mathcal{F}_{\text{PAKE}}$ honestly, and samples a random string as h, unless P's instance has been successfully attacked — in which case $\mathcal{S}$ knows P's EKE key and can simulate h honestly:

Simulator $\mathcal{S}$:

0. Answer H and H_0 queries (either from $\mathcal{A}$ or from $\mathcal{S}$ itself) via lazy sampling.

1. On $(\mathsf{NewSession}, sid, P, P')$ from $\mathcal{F}$, send the same message from $\mathcal{F}_{\text{PAKE}}$ to $\mathcal{A}$.
   On $(\mathsf{NewSession}, sid, P', P)$ from $\mathcal{F}$, also send the same message from $\mathcal{F}_{\text{PAKE}}$ to $\mathcal{A}$.

2. On $(\mathsf{TestPwd}, sid, \star, \star)$ from $\mathcal{A}$ to $\mathcal{F}_{\text{PAKE}}$, forward this message to $\mathcal{F}$ and sends $\mathcal{F}$'s responde from $\mathcal{F}_{\text{PAKE}}$ to $\mathcal{A}$.

3. On $(\mathsf{NewKey}, sid, P, K^*)$ from $\mathcal{A}$ to $\mathcal{F}_{\text{PAKE}}$,

   (i) If there has been a $(\mathsf{TestPwd}, sid, P, \star)$ command from $\mathcal{A}$ to $\mathcal{F}_{\text{PAKE}}$ which resulted in "correct guess" (see step 2), then set $h := H_0(K^*)$; otherwise sample $h \leftarrow \{0,1\}^\lambda$. Either way, send h from P to P'.

   (ii) Also compute $SK^* := H(K^*)$ and send $(\mathsf{NewKey}, sid, P, SK^*)$ to $\mathcal{F}$.

4. On $(\mathsf{NewKey}, sid, P', (K')^*)$ from $\mathcal{A}$ to $\mathcal{F}_{\text{PAKE}}$ and h^* from $\mathcal{A}$ to P', compute $(SK')^* := H((K')^*)$ and send $(\mathsf{NewKey}, sid, P', (SK')^*)$ to $\mathcal{F}$.

We now show that for any PPT environment $\mathcal{Z}$, $\mathcal{S}$ generates an ideal-world view that is indistinguishable from $\mathcal{Z}$'s real-world view. Let K and K' be P and P''s EKE key, respectively, and SK and SK' be P and P''s final output key in $\tilde{\Pi}$, respectively. Define QueryP as the event that

- $\mathcal{A}$ does not send a $(\mathsf{TestPwd}, sid, P, \star)$ command resulting in "correct guess", but
- $\mathcal{A}$ queries $H_0(K)$ or $H(K)$.

Define QueryP' as the event that

- $\mathcal{A}$ does not send a $(\mathsf{TestPwd}, sid, P', \star)$ command resulting in "correct guess", but
- $\mathcal{A}$ queries $H(K')$.

Claim 1. $\Pr[\mathsf{QueryP}]$ *and* $\Pr[\mathsf{QueryP}']$ *are both negligible.*

If $\mathcal{A}$ does not send a $(\mathsf{TestPwd}, sid, P, \star)$ command resulting in "correct guess", then $\mathcal{F}_{\text{PAKE}}$ marks P's session as either fresh or $\mathsf{interrupted}$. Either way, when K is generated it is a uniformly random string in $\{0,1\}^\lambda$ independent of $\mathcal{Z}$'s view, and later the only information $\mathcal{Z}$ learns about K is $H_0(K)$ (from the protocol message) and $H(K)$ (from P's output key) — which are independent of K unless and until QueryP happens. Therefore, $\Pr[\mathsf{QueryP}]$ is upper-bounded by $q_H/2^\lambda$ (where q_H is the total number of H and H_0 queries). A similar argument works for QueryP'.

Claim 2. *If neither* $\Pr[\mathsf{QueryP}]$ *nor* $\Pr[\mathsf{QueryP}']$ *happens, then* $\mathcal{Z}$*'s views in the real world and in the ideal world are identical.*

The argument goes as follows:

1. Regarding $\mathcal{A}$'s interface with $\mathcal{F}_{\text{PAKE}}$, $\mathcal{S}$ does nothing other than forwarding $\mathcal{A}$'s message to $\mathcal{F}$ and forwarding $\mathcal{F}$'s response back to $\mathcal{A}$. In other words, $\mathcal{S}$ simulates $\mathcal{F}_{\text{PAKE}}$ honestly. So $\mathcal{S}$ generates a view that is identical to the real-world view.

2. Regarding protocol message h, first note that in both worlds this message is generated right after $\mathcal{A}$ sends $(\mathsf{NewKey}, sid, P, K^*)$ to $\mathcal{F}_{\text{PAKE}}$.

   - If $\mathcal{A}$ has sent a $(\mathsf{TestPwd}, sid, P, \star)$ command resulting in "correct guess", then in the real world P's instance in $\mathcal{F}$ was $\mathsf{compromised}$. This causes P's EKE key K to be equal to K^*, so $h = H_0(K) = H_0(K^*)$. This exactly matches how h is generated by $\mathcal{S}$.
   - Otherwise, we know that $\mathcal{A}$ does not query $H_0(K)$ (because QueryP does not happen). This means that in the real world $h = H_0(K)$ is a uniformly random string in $\{0,1\}^\lambda$ independent of the rest of the game, which again matches how h is generated by $\mathcal{S}$.

3. Regarding P's output key SK,

   - If $\mathcal{A}$ has sent a $(\mathsf{TestPwd}, sid, P, \star)$ command resulting in "correct guess", then as argued above, in the real world $K = K^*$, so $SK = H(K) = H(K^*)$. In the ideal world, $\mathcal{S}$ computes $SK^* = H(K^*)$, so $SK^* = SK$. Furthermore, $\mathcal{F}$ has marked P's instance $\mathsf{compromised}$ (because $\mathcal{S}$ has sent a $(\mathsf{TestPwd}, sid, P, \star)$ command resulting in "correct guess"), so when $\mathcal{S}$ sends $(\mathsf{NewKey}, sid, P, SK^*)$ to $\mathcal{F}$, $\mathcal{F}$ outputs SK^* to P. In sum, in both worlds P's output key is $SK = H(K^*)$, so there is no difference.
   - Otherwise, we know that $\mathcal{A}$ does not query $H(K)$ (because QueryP does not happen). This means that in the real world SK is a uniformly random string in $\{0,1\}^\lambda$ independent of the rest of the game. In the ideal world, $\mathcal{F}$ has marked P's instance fresh or $\mathsf{interrupted}$, so when $\mathcal{S}$ sends $(\mathsf{NewKey}, sid, P, SK^*)$ to $\mathcal{F}$, $\mathcal{F}$ ignores SK^* and freshly samples a uniformly random string in $\{0,1\}^\lambda$ as P's output key. In sum, in both worlds P's output key is $SK \leftarrow \{0,1\}^\lambda$, so there is no difference.

   The same argument can be made for P''s output key SK'.

Combining the two claims above immediately yields the UC-security of $\tilde{\Pi}$. □

3.3 The Protocol Is Not KOY-Secure

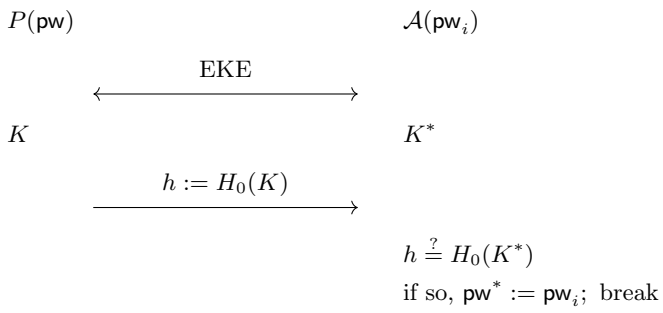
Proof. Consider the following adversary $\mathcal{A}$. Roughly speaking, $\mathcal{A}$ assumes the role of P' and runs P''s algorithm on all candidate passwords, and uses h to find out which one is the correct password (after which the attack becomes trivial):

Adversary $\mathcal{A}$:

For all $\mathsf{pw}_i \in D$, $\mathcal{A}$ performs the following steps until it finds pw^*:

1. Initiate P's instance P^i by sending $\mathsf{Send}(P^i, (P')^j, \text{start})$ to the game challenger.
2. On c from the game challenger, sample $y \leftarrow \mathbb{Z}_p$ and
   (i) Compute $Y^* := \mathcal{E}(\mathsf{pw}_i, g^y)$ and send $\mathsf{Send}((P')^j, P^i, Y^*)$ to the game challenger.
   (ii) Also, compute $X := \mathcal{D}(\mathsf{pw}_i, c)$ and then $K^* := X^y$.
3. On h from the game challenger, check if $h = H_0(K^*)$. If so, set $\mathsf{pw}^* := \mathsf{pw}_i$ and break the for-loop. Otherwise move on to the next element in D.

See below for a graphic illustration of steps 1–3 of $\mathcal{A}$.



After the for-loop ends, $\mathcal{A}$ finds the i such that $\mathsf{pw}^* = \mathsf{pw}_i$ and performs the following step:

4. Send $\mathsf{Test}(P^i)$ to the game challenger. If the result is equal to $H(K^*)$, output 1; otherwise output 0.

Note that $\mathcal{A}$ sends no Reveal command and a single Test command, so it only does one online attack and its "baseline" winning probability is $1/2 + 1/2|D|$. However, we show that $\mathcal{A}$'s winning probability is in fact overwhelming:

Claim 3. *The probability that* $\mathsf{pw}^* = \mathsf{pw}$*, i.e.,* $\mathcal{A}$ *finds the correct password, is overwhelming.*

EKE has the property that the two parties' output keys are equal *if and only if* their passwords match. Note that $\mathcal{A}$'s steps 1–2 are exactly what an honest P' would to on password pw_i; therefore, if $\mathsf{pw}_i \neq \mathsf{pw}$ then P's EKE key $K \neq K^*$, so $h = H_0(K)$ is not equal to $H_0(K^*)$ except with probability $1/2^\lambda$, causing $\mathcal{A}$ to move on to the next candidate password. If $\mathsf{pw}_i = \mathsf{pw}$ then P's EKE key $K = K^*$, so $h = H_0(K) = H_0(K')$, causing $\mathcal{A}$ to break the for-loop and set $\mathsf{pw}^* := \mathsf{pw}_i = \mathsf{pw}$. Overall, $\mathcal{A}$ sets $\mathsf{pw}^* = \mathsf{pw}$ except with probability up to $(|D| - 1)/2^\lambda$.

Claim 4. *If* $\mathsf{pw}^* = \mathsf{pw}$*, then* $\mathcal{A}$ *wins with overwhelming probability.*

Suppose $b = 1$. Then when $\mathcal{A}$ sends $\mathsf{Test}(P^i)$, the game challenger returns P^i's output key $H(K)$. But if $\mathsf{pw}^* = \mathsf{pw}$ then $K^* = K$, so $H(K) = H(K^*)$ and $\mathcal{A}$ outputs 1.

Now suppose $b = 0$. Then when $\mathcal{A}$ sends $\mathsf{Test}(P^i)$, the game challenger returns a uniformly random string in $\{0,1\}^\lambda$ independent of the rest of the game. Therefore, $\mathcal{A}$ outputs 1 with probability $1/2^\lambda$.

In sum, $\mathcal{A}$ wins with probability 1 when $b = 1$ and $1 - 1/2^\lambda$ when $b = 0$. Its overall winning probability is $1 - 1/2^{\lambda+1}$.

Combining the two claim above immediately yields that $\mathcal{A}$ wins with overwhelming probability, so Π is not KOY-secure.[7] □

Π is still BPR-secure. Our attack above does not violate the BPR-security of Π. This is because the sentence in blue does not hold anymore: $\mathcal{A}$ sends up to $|D|$ Send commands, so in BPR the number of online attacks is up to $|D|$ and thus $\mathcal{A}$'s "baseline" winning probability is $1/2 + |D|/2|D| = 1$.

4 Implicit-Only UC-Security Implies KOY-Security

Our main result in this section is:

Theorem 1. *Suppose* Π *is a PAKE protocol that UC-realizes* $\mathcal{F}_{\text{iPAKE}}$*. Then* Π *is KOY-secure.*

We need the following technical lemma from [CHK+05, Lemma A.1] (for a detailed proof, see [RX23, Appendix A]):

Lemma 1. *Suppose* Π *is a PAKE protocol that UC-realizes* $\mathcal{F}_{\text{iPAKE}}$*. Fix any "successful" simulator* $\mathcal{S}$ *that, for any PPT environment, generates a view indistinguishable from the environment's real-world view in* Π*; and any PPT environment* $\mathcal{Z}$*. Let* $\mathsf{SpuriousGuess}$ *be the event that there is a session sid in which* $\mathcal{Z}$ *instructs the adversary to be passive, yet* $\mathcal{S}$ *sends* $(\mathsf{TestPwd}, sid, \star, \star)$ *to* $\mathcal{F}_{\text{iPAKE}}$*. Then* $\Pr[\mathsf{SpuriousGuess}] \leq \mathsf{negl}$.[8]

We are now ready to prove Theorem 1.

Proof. Consider any PPT adversary $\widehat{\mathcal{A}}$ in the KOY-security game for Π, and let q be the number of online attacks by $\widehat{\mathcal{A}}$; we want to upper-bound $\widehat{\mathcal{A}}$'s winning probability. (We use $\widehat{\mathcal{A}}$ instead of $\mathcal{A}$ to emphasize that it is the adversary in the game-based definition, not the UC adversary.) The outline of the proof is as follows:

[7] Here we need $|D| \geq 2$; otherwise $\mathcal{A}$'s "baseline" winning probability is 1, so winning with overwhelming probability does not violate the security definition.

[8] Technically speaking Π must be correct, i.e., if there is no adversary then the two protocol parties output the same key. See [RX23] for a discussion. In this work we always assume PAKE protocols to be correct.

1. We first "translate" $\widehat{\mathcal{A}}$ into a UC environment $\mathcal{Z}$ that essentially does what $\widehat{\mathcal{A}}$ does during protocol execution. $\mathcal{Z}$ outputs 1 if and only if $\widehat{\mathcal{A}}$ wins.
2. Since $\mathcal{Z}$ simulates the KOY-security game for $\widehat{\mathcal{A}}$, the probability that $\mathcal{Z}$ outputs 1 in the real world is equal to $\widehat{\mathcal{A}}$'s winning probability.
3. However, in the ideal world $\widehat{\mathcal{A}}$'s winning probability is limited by how much it learns about the password. But $\widehat{\mathcal{A}}$ gains some information about the password only if *both* (1) the simulator does a successful online attack, and (2) $\widehat{\mathcal{A}}$ sees the attacked instance's output key. The number of occurrences of (1) and (2) is exactly q.
4. Therefore, $\widehat{\mathcal{A}}$'s winning probability must be close to the "baseline".

Steps 1 and 2 are essentially identical to the proof of UC-security $\Rightarrow$ BPR-security [CHK+05]; the main technical challenge lies in step 3.

"Translating" $\widehat{\mathcal{A}}$ into a UC environment. Consider the following UC environment $\mathcal{Z}$:

Environment $\mathcal{Z}$:

0. Sample $b \leftarrow \{0,1\}$ and invoke $\widehat{\mathcal{A}}$.
1. For each pair of parties (P, P'), sample a password $\mathsf{pw} \leftarrow D$.
2. On $\mathsf{Execute}(P^i, (P')^j)$ from $\widehat{\mathcal{A}}$, pick a fresh $sid_{i,j} = sid_{j,i}$ and send $(\mathsf{NewSession}, sid_{i,j}, P', \mathsf{pw})$ to P' and $(\mathsf{NewSession}, sid_{i,j}, P, \mathsf{pw}_{i,j})$ to P. (Below we may simply write sid when i, j are clear from the context.) Then instruct the UC PAKE adversary $\mathcal{A}$ to be passive in this session. After the session completes, provide $\widehat{\mathcal{A}}$ with its transcript. (Note that P^i is an instance in the KOY-security game simulated by $\mathcal{Z}$, whereas P is a UC protocol party that interacts with $\mathcal{Z}$.)
3. On $\mathsf{Send}(P^i, (P')^j, \text{start})$ from $\widehat{\mathcal{A}}$, if there has been a $\mathsf{Send}((P')^j, P^i, \text{start})$ command, then $sid_{j,i} = sid_{i,j}$ must have been defined. In this case, send $(\mathsf{NewSession}, sid_{i,j}, P', \mathsf{pw})$ to P. Otherwise pick a fresh $sid_{i,j} = sid_{j,i}$ and send $(\mathsf{NewSession}, sid_{i,j}, P', \mathsf{pw})$ to P.
   On $\mathsf{Send}(P^i, (P')^j, m^*)$ from $\widehat{\mathcal{A}}$, instruct the UC PAKE adversary $\mathcal{A}$ to send a message (sid, m^*) to P'.
   In all cases above, forward the message $\mathcal{A}$ receives to $\widehat{\mathcal{A}}$.
4. On $\mathsf{Reveal}(P^i)$ from $\widehat{\mathcal{A}}$, fetch $sid_{i,j}$ (where j is the index of instance $(P')^j$ that P^i communicates with). If P has output $(sid_{i,j}, SK)$ to $\mathcal{Z}$, return SK; otherwise return $\perp$.
5. $\mathsf{Test}(P^i)$ from $\widehat{\mathcal{A}}$ is handled in the same way as $\mathsf{Reveal}(P^i)$, except that "return SK" is replaced by "return SK if $b = 1$ and a random string in $\{0,1\}^\lambda$ if $b = 0$".
6. When $\widehat{\mathcal{A}}$ outputs a bit b^*, output 1 if $b^* = b$ and 0 otherwise.

Analysis in the real world. If $\mathcal{Z}$ is in the real world, then it exactly simulates Π's KOY-security game for $\widehat{\mathcal{A}}$. (The reader should check that this is indeed the case.) We have

$$\Pr[\mathcal{Z} \text{ outputs 1 in the real world}] = \Pr[b^* = b \text{ in the real world}] = \Pr[\widehat{\mathcal{A}} \text{ wins}].$$

Analysis in the ideal world. Now suppose $\mathcal{Z}$ is in the ideal world. Let $\mathcal{S}$ be a simulator that, for any PPT environment, generates an ideal-world view indistinguishable from the environment's real-world view. We now upper-bound $\Pr[b^* = b \text{ in the ideal world}]$.

Define "bad event"

$$\begin{aligned}\mathsf{CorrectGuess} : \mathcal{S} \text{ sends } (\mathsf{TestPwd}, sid_{i,j}, P, \mathsf{pw}) \text{ to } \mathcal{F}_{\text{iPAKE}} \\ \text{where } \widehat{\mathcal{A}} \text{ performs an online attack on } P^i,\end{aligned}$$

i.e., CorrectGuess happens if $\mathcal{S}$ tests the correct password pw on an instance on which $\widehat{\mathcal{A}}$ performs an online attack. (Intuitively, CorrectGuess happens if $\widehat{\mathcal{A}}$'s protocol messages contain password guess pw, which is extracted by $\mathcal{S}$; and eventually $\widehat{\mathcal{A}}$ learns this instance's output key.)

Claim 5. *Suppose* SpuriousGuess *does not happen. Then* $\Pr[\mathsf{CorrectGuess}] \leq \frac{q}{|D|}$.

This claim is the crux of the entire proof (and where we crucially deviate from [CHK+05]). The key point is to *view* $\widehat{\mathcal{A}}$ *and* $\mathcal{S}$ *combined as a single entity, and see what information it receives about* pw. In fact there is only a single cycle through which $\widehat{\mathcal{A}}$ and $\mathcal{S}$ might (jointly) learn some information about pw:

1. For a certain instance, $\mathcal{S}$ submits a password guess pw^* (via TestPwd) and a key SK^* (via NewKey) to $\mathcal{F}_{\text{iPAKE}}$.
2. $\mathcal{F}_{\text{iPAKE}}$ generates key SK which is equal to SK^* if $\mathsf{pw}^* = \mathsf{pw}$ and uniformly random otherwise. $\mathcal{F}_{\text{iPAKE}}$ then sends SK to the UC protocol party P.
3. P forwards SK to $\mathcal{Z}$.
4. When $\widehat{\mathcal{A}}$ sends a Reveal command for this instance, or a Test command in the case of $b = 1$, $\mathcal{Z}$ returns SK to $\widehat{\mathcal{A}}$.

(Other information $\widehat{\mathcal{A}}$ and $\mathcal{S}$ may receive is NewSession commands from $\mathcal{F}_{\text{iPAKE}}$ to $\mathcal{S}$, indicating that an instance has been initiated. This message is of course independent of pw. Note in particular that protocol messages $\widehat{\mathcal{A}}$ receives on Execute and Send commands are in fact simulated by $\mathcal{S}$: upon receiving such a command, $\mathcal{Z}$ lets $\mathcal{S}$ simulate protocol messages for the UC PAKE adversary $\mathcal{A}$, which $\mathcal{Z}$ forwards to $\widehat{\mathcal{A}}$. Therefore, when we view $\widehat{\mathcal{A}}$ and $\mathcal{S}$ jointly, Execute and Send commands (without Reveal or Test) provide no information about pw.)

Let's call steps 1–4 above a "cycle". In short, the game becomes: in each cycle $\{\widehat{\mathcal{A}}, \mathcal{S}\}$ may specify $\mathsf{pw}^* \in D$ and SK^*, and receive SK^* if $\mathsf{pw}^* = \mathsf{pw}$ and a uniformly random SK otherwise. CorrectGuess is triggered when $\{\widehat{\mathcal{A}}, \mathcal{S}\}$ specifies pw. In each cycle the only information $\{\widehat{\mathcal{A}}, \mathcal{S}\}$ learns about pw is whether $\mathsf{pw}^* = \mathsf{pw}$, so Pr[CorrectGuess] is upper-bounded by (number of cycles)/$|D|$.

How many cycles can $\{\widehat{\mathcal{A}}, \mathcal{S}\}$ complete? First, $\widehat{\mathcal{A}}$ must send a Reveal or a Test command on this instance (otherwise $\{\widehat{\mathcal{A}}, \mathcal{S}\}$ does not learn SK and thus learns no information about pw). Second, if SpuriousGuess does not happen then this instance must be a Send instance (otherwise $\mathcal{S}$ does not even send a TestPwd command, i.e., $\{\widehat{\mathcal{A}}, \mathcal{S}\}$ does not even specify pw^*). But a cycle that satisfies these conditions is exactly an online attack. Therefore, the number of cycles is at most q.

We conclude that

$$\Pr[\mathsf{CorrectGuess}] \leq \frac{\text{number of cycles}}{|D|} \leq \frac{q}{|D|}.$$

Claim 6. *Suppose again* SpuriousGuess *does not happen. If* CorrectGuess *also does not happen, then $b^* = b$ with probability $1/2$.*

Consider the instance for which $\widehat{\mathcal{A}}$ sends Test. If it is an Execute instance, then $\mathcal{S}$ does not send TestPwd (because SpuriousGuess does not happen), so this session is fresh and outputs a uniformly random key. If it is a Send instance, then this instance is actively attacked, so $\mathcal{S}$ does not send TestPwd on the correct password (because CorrectGuess does not happen), so this session is fresh or interrupted and outputs a uniformly random key. This means that b is independent of $\widehat{\mathcal{A}}$'s view, so $\Pr[b^* = b] = 1/2$.

Combining the two claims above, we get

$$\begin{aligned}
&\Pr[\mathcal{Z} \text{ outputs 1 in the ideal world}] \\
\leq &\Pr[\mathsf{CorrectGuess} \mid \overline{\mathsf{SpuriousGuess}}] + \Pr[b^* = b \wedge \overline{\mathsf{CorrectGuess}} \mid \overline{\mathsf{SpuriousGuess}}] + \Pr[\mathsf{SpuriousGuess}] \\
\leq &\Pr[\mathsf{CorrectGuess} \mid \overline{\mathsf{SpuriousGuess}}] + \frac{1}{2}(1 - \Pr[\mathsf{CorrectGuess} \mid \overline{\mathsf{SpuriousGuess}}]) + \Pr[\mathsf{SpuriousGuess}] \\
= &\frac{1}{2} + \frac{1}{2}\Pr[\mathsf{CorrectGuess} \mid \overline{\mathsf{SpuriousGuess}}] + \Pr[\mathsf{SpuriousGuess}] \\
\leq &\frac{1}{2} + \frac{q}{2|D|} + \Pr[\mathsf{SpuriousGuess}] \\
\leq &\frac{1}{2} + \frac{q}{2|D|} + \mathsf{negl},
\end{aligned}$$

where the last inequality is due to Lemma 1.

Putting it together. In sum, $\mathcal{Z}$'s distinguishing advantage is at least

$$\Pr[\widehat{\mathcal{A}} \text{ wins}] - \frac{1}{2} - \frac{q}{2|D|} - \mathsf{negl},$$

which is negligible since Π UC-realizes $\mathcal{F}_{\text{iPAKE}}$. So

$$\Pr[\widehat{\mathcal{A}} \text{ wins}] \leq \frac{1}{2} + \frac{q}{2|D|} + \mathsf{negl},$$

completing the proof. □

Recovering the UC-security ⇒ BPR-security proof. One might ask where the proof breaks down if Π only realizes $\mathcal{F}_{\text{PAKE}}$. The reason is that the statement in blue does not hold anymore, as there is another cycle that allows the combination of $\widehat{\mathcal{A}}$ and $\mathcal{S}$ to learn some information about pw:

1. For a certain instance, $\mathcal{S}$ submits a password guess pw^* (via $\mathsf{TestPwd}$) to $\mathcal{F}_{\text{PAKE}}$.
2. $\mathcal{F}_{\text{PAKE}}$ returns "correct/wrong guess" to $\mathcal{S}$.

Crucially, this cycle does not require $\widehat{\mathcal{A}}$ to send Reveal or Test, i.e., $\mathcal{S}$ can "help" $\widehat{\mathcal{A}}$ gain some information about pw without $\widehat{\mathcal{A}}$ performing an online attack (per the KOY notion). This is exactly what our counterexample in Section 3 exploits.

On the other hand, if we take the BPR notion (where a Send command already counts as an online attack), then the cycle above still requires an online attack (assuming $\mathsf{SpuriousGuess}$ does not happen) and the rest of the proof still goes through. In this way we recover the security proof of UC-security ⇒ BPR-security [CHK+05].

5 Discussion

How to interpret our results? Let's first look at the difference between the two game-based security definitions. In BPR-security, any Send command to an instance counts as an online attack; whereas in KOY-security, an online attack must also Reveal or Test the instance's key. In other words, BPR-security allows the protocol transcript alone to leak some information about the password in an active attack, whereas in KOY-security only the protocol transcript *plus the attacked instance's output key* may tell the adversary something about the password — which is inevitable.[9] Our counterexample in Section 3 exactly exploits this: the protocol message h allows the adversary to check password guesses — and eventually find the correct password — by sending Send commands only, violating KOY- but not BPR-security.

The difference between UC- and implicit-only UC-security can be interpreted in a similar way. Consider the two ideal functionalities: $\mathcal{F}_{\text{PAKE}}$ allows the ideal adversary to check if its password guess is correct or not via a $\mathsf{TestPwd}$ command, which means that in the real world the adversary can learn this bit of information by observing the transcript in an active attack; whereas in $\mathcal{F}_{\text{iPAKE}}$

[9] Note that even in BPR-security, the transcript should not leak any information about the password if the adversary is passive (i.e., the adversary sends an $\mathsf{Execute}$ command).

the adversary does not learn this bit. In fact, in $\mathcal{F}_{\text{iPAKE}}$ *the ideal adversary never learns any information about any instance's password*, which is crucial in our proof in Section 4.

Our conclusion is that the KOY- and implicit-only UC-security notions strengthen their standard counterparts in the same manner, namely

> *KOY- and implicit-only UC-security disallow the adversary from learning whether an online attack has succeeded or not by merely examining the protocol transcript (without seeing the attacked instance's output key).*

We now explain why this property is called "implicit-only". PAKE with explicit authentication allows each protocol party to check whether the authentication has been successful, i.e., whether its output key is equal to its counterparty's. But if an honest party can learn this, then an adversary that impersonates this party on a password guess can also learn this; that is, an adversary that communicates with Alice by running Bob's algorithm on password guess pw^*, can learn whether it computes the same key as Alice's output key — which in turn allows the adversary to check if pw^* is equal to Alice's password. In this way, the adversary can check a password guess without seeing Alice's output key, violating KOY- or implicit-only UC-security. In other words, *a PAKE protocol with explicit authentication cannot be implicit-only UC-secure* — indeed, the message h in our counterexample is exactly what a protocol party would send if we wanted to add explicit authentication to EKE.

At this point one may wonder whether this "implicit-only" property is a feature or a bug, or which definition is the "right" one. This depends on the exact security notion one wants to achieve: as discussed above, if an honest party can learn more information (explicit authentication — feature), then the adversary can also learn this information (the "correct/wrong" guess bit — bug), so the feature and the bug must appear simultaneously. As such, the two (sets of) definitions model slightly different properties of the protocol: one has stronger honest parties and a stronger adversary, whereas the other has weaker honest perties and a weaker adversary. While the BPR- and UC-security notions are generally considered to be the "standard", we believe that KOY- and implicit-only UC-security in fact captures the security level of a large number of PAKE protocols more accurately and deserves more attention from the community.

References

[BM92] Steven M. Bellovin and Michael Merritt. Encrypted key exchange: Password-based protocols secure against dictionary attacks. In *S&P 1992*, pages 72–84, 1992.

[BPR00] Mihir Bellare, David Pointcheval, and Phillip Rogaway. Authenticated key exchange secure against dictionary attacks. In *EUROCRYPT 2000*, pages 139–155, 2000.

[Can01] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *FOCS 2001*, pages 136–145, 2001.

[CHK+05] Ran Canetti, Shai Halevi, Jonathan Katz, Yehuda Lindell, and Philip D. MacKenzie. Universally composable password-based key exchange. In *EUROCRYPT 2005*, pages 404–421, 2005.

[DHP+18] Pierre-Alain Dupont, Julia Hesse, David Pointcheval, Leonid Reyzin, and Sophia Yakoubov. Fuzzy password-authenticated key exchange. In *EUROCRYPT 2018*, pages 393–424, 2018.

[JRX25] Jake Januzelli, Lawrence Roy, and Jiayu Xu. Under what conditions is encrypted key exchange actually secure? In *EUROCRYPT 2025*, pages 451–481, 2025.

[KOY01] Jonathan Katz, Rafail Ostrovsky, and Moti Yung. Efficient password-authenticated key exchange using human-memorable passwords. In *EUROCRYPT 2001*, pages 475–494, 2001.

[KOY09] Jonathan Katz, Rafail Ostrovsky, and Moti Yung. Efficient and secure authenticated key exchange using weak passwords. *Journal of the ACM*, 57(1):78–116, 2009.

[KV11] Jonathan Katz and Vinod Vaikuntanathan. Round-optimal password-based authenticated key exchange. In *TCC 2011*, pages 293–310, 2011.

[RX23] Lawrence Roy and Jiayu Xu. A universally composable PAKE with zero communication cost (And why it shouldn't be considered UC-secure). In *PKC 2023*, pages 714–743, 2023.

[Xu25] Jiayu Xu. UC-security of encrypted key exchange: A tutorial. Cryptology ePrint Archive, 2025.