

UC PAKE and Concrete Security

Tingfei Feng¹ and Jiayu Xu²

¹Oregon State University, fengti@oregonstate.edu

²Oregon State University, xujiay@oregonstate.edu

Abstract

The Universal Composability (UC) framework provides a paradigm for security definitions that allows for arbitrary composition, but has been criticized for being unamicable to the concrete security approach, as the meaning of a concrete UC-security statement is hard to interpret in practice. This work aims at alleviating this concern in the context of Password-Authenticated Key Exchange (PAKE): we show that any concrete UC-security statement for PAKE can be easily converted to a concrete game-based security statement, whose real-life meaning is clearer. We also briefly discuss potential generalizations of our result to other cryptographic primitives.

1 Introduction

Universal Composability (UC), introduced by Canetti in 2001 [5], remains a widely popular framework for security definitions after 25 years. Compared with traditional security definitions in the game-based setting, UC has the advantage of preserving security under arbitrary composition, either with the same protocol or with other protocols, sequentially or concurrently. On the other hand, UC suffers from a few drawbacks: for example, ideal functionalities are sometimes hard to understand and easy to misunderstand [1], and the real-life implication of UC-insecurity is sometimes unclear [8, Section 5].

This work focuses on another, more recent criticism of UC, namely that it is unamicable to the concrete security approach. Indeed, in the game-based setting, (T, ϵ) -security can be easily interpreted as “an adversary that runs in time T can break the protocol with advantage at most ϵ ”, whereas a concrete security statement in UC has to involve three parameters: the runtime of the simulator T_S , the runtime of the environment T , and the environment’s distinguishing advantage ϵ . (This is assuming that the real-world adversary is “dummy”, which is standard in UC.) Since T_S may affect the values of T and ϵ , it is unclear what this statement actually means in real life; in particular, even with such a statement, we still have little idea on how to choose the security parameter.

This issue has (somehow) been ignored by researchers who define and prove security in UC. As acutely pointed out in [4, Section 1.4],

There is a divide, in the cryptographic community, between those who speak and use the language of UC, and those who don't. A consequence has been to exclude a certain, and more applied part of our community, from 2PC research.

This work aims to bridge the gap between UC-security and concrete security, using *Password-Authenticated Key Exchange (PAKE)* as a case study. We believe PAKE is a topic uniquely suitable for our purpose: the standard security notion for PAKE is in UC, as demonstrated by, e.g., the fact that all four PAKE protocols that made it to the second round of the 2019–20 IETF standardization competition [7] have a UC-security proof; on the other hand, since PAKE has the potential of being widely used in practice, concrete security analyses — which directly guides the choice of the security parameter — are urgently needed.

Fortunately, the “research infrastructure” for our task is already in place. Both the standard game-based security for PAKE by Bellare, Pointcheval and Rogaway (BPR) [3], and the standard UC functionality for PAKE [6], have been widely studied and accepted by the community; furthermore, a standard “UC implies game-based” theorem has already been proven [3, Appendix A] — *albeit in the asymptotic setting*. This work can be viewed as restating and reproving [3, Appendix A] in the concrete security setting; roughly speaking, our main result is:

Theorem 1 (informal; see Section 3.2 for a formal statement). *Suppose Π (T, ϵ) -realizes the PAKE functionality, namely any UC environment that runs in time T has distinguishing advantage at most ϵ . Then Π is roughly $(T, q\epsilon)$ -BPR-secure, where q is the number of instances that the BPR adversary initiates.*

While the proof is technically straightforward, our result helps clarify a few important conceptual points. First, given *any* concrete security statement for UC PAKE, we can immediately translate it into a concrete game-based security statement; as such, the “UC is unamicable to concrete security analyses” criticism is invalid in the context of PAKE (after our work). Second, note that the theorem statement above does not involve the runtime of the simulator T_S ; this opens the door to interpreting a concrete UC-security statement (and thus choosing the security parameter) directly, without resorting to a game-based security statement. Third, the fact that the theorem does not rely on T_S also means that a “pseudo-UC-security” statement, where the simulator is computationally unbounded, is sufficient for game-based security, in the asymptotic and concrete security settings alike.¹

We caution that while Theorem 1 does not depend on the simulator’s runtime T_S , it is possible — and indeed often the case — that the environment’s

¹The last point was first observed in the context of *asymmetric* PAKE in [12, Section 6], although their conclusion is in the asymptotic setting only.

distinguishing advantage ϵ does depend on T_S . Indeed, ϵ is usually derived from some reduction $\mathcal{R}$'s advantage, and $\mathcal{R}$ may essentially run the code of the simulator (plus the UC PAKE functionality); therefore, ϵ may depend on $\mathcal{R}$'s advantage, which in turn depends on $\mathcal{R}$'s runtime, which may be related to T_S . (This observation is made in [2].) However, in a concrete security proof via the chain

$$\text{cryptographic assumption} \Rightarrow \text{UC-security} \Rightarrow \text{BPR-security},$$

we might as well calculate the runtime of the reduction(s) $\mathcal{R}$ directly, without explicitly considering T_S .²

The rest of this work is organized as follows. In Section 2 we review the necessary preliminaries, including the concrete security definitions for PAKE in the game-based and UC settings. In Section 3 we present and prove our main theorem. Finally, in Section 4 we discuss (1) the implication of our result on a recent work, and (2) some potential future directions.

2 Preliminaries

We use λ to denote the security parameter.

2.1 BPR-Security Definition for PAKE

In this section we review the BPR-security definition for PAKE, rendered in the concrete security setting. We mostly follow the presentation in [11, Section 2], and are satisfied with merely showing the technical details of the definition; for an intuitive explanation, see [11].

Initialization. The system consists of multiple protocol parties P , and each party may run multiple instances P^i . Instances belonging to different parties can exchange protocol messages with one another. Let Dict denote the public set of all possible passwords; we require that $|\text{Dict}| \geq 2$ and that sampling an element from Dict can be done in time $\mathcal{O}(\lambda)$. For each pair of parties $P \neq P'$, sample $\text{pw} \leftarrow \text{Dict}$ as the common password shared by them.

Protocol execution. The BPR-security of PAKE is defined as a game between an adversary $\mathcal{A}$ and a challenger. The challenger runs the protocol on behalf of the parties and their instances, and reveal to $\mathcal{A}$ the information that $\mathcal{A}$ can learn during different forms of attacks. The following commands are the oracle calls that $\mathcal{A}$ can issue to the challenger:

- On $\text{Execute}(P^i, (P')^j)$: If P^i and $(P')^j$ are both unused, the challenger executes the protocol between P^i and $(P')^j$ and forwards the transcript to $\mathcal{A}$, and mark P^i and $(P')^j$ as used.

²It is not immediately clear to what extent $\mathcal{R}$'s runtime depends on T_S , since $\mathcal{R}$ usually simulates some intermediate hybrid rather than the ideal world. It appears to us that in most cases $\mathcal{R}$'s runtime is more closely tied to the environment's runtime T than to T_S .

- On $\text{Send}(P^i, (P')^j, m^*)$: The challenger sends m^* on behalf of the instance P^i to $(P')^j$. If $(P')^j$ sends back a message m' , the challenger forwards m' to $\mathcal{A}$; otherwise the challenger sends $\perp$ to $\mathcal{A}$, as the instance $(P')^j$ halts.
If $m^* = \text{start}$, as an indication to initiate the instance: if P^i is not used as an instance before in Send or Execute , the challenger initiates P^i and forwards P^i 's message to $\mathcal{A}$, and marks the instance P^i as used.
- On $\text{Reveal}(P^i)$: If P^i has computed its session key K_i , the challenger forwards K_i to $\mathcal{A}$; otherwise the challenger sends $\perp$ to $\mathcal{A}$.
- On $\text{Test}(P^i)$: The challenger samples a bit $b \leftarrow \{0, 1\}$. If P^i has computed its session key K_i , the challenger sends K_i to $\mathcal{A}$ if $b = 1$, or sends a uniformly random string in $\{0, 1\}^\lambda$ if $b = 0$; otherwise the challenger sends $\perp$ to $\mathcal{A}$.

At the end of the game, $\mathcal{A}$ outputs a bit b^* and wins if $b^* = b$. Before we define the security by bounding the winning probability of $\mathcal{A}$, we first define the following events that allow $\mathcal{A}$ to win trivially:

- If $\mathcal{A}$ ever calls $\text{Reveal}(P^i)$ and $\text{Test}(P^i)$ for the same P^i .
- If $\mathcal{A}$ calls $\text{Execute}(P^i, (P')^j)$, and either calls $\text{Reveal}(P^i)$ and $\text{Test}((P')^j)$, or calls $\text{Test}(P^i)$ and $\text{Reveal}((P')^j)$.
- If $\mathcal{A}$ calls Send on P^i and $(P')^j$, but P^i and $(P')^j$ share an identical view. This can be achieved by having $\mathcal{A}$ forward each output message from one instance as the corresponding input message to the other instance.

We are now ready to present the BPR-security definition:

Definition 1. A PAKE protocol is (T, ϵ) -BPR-secure if for any adversary $\mathcal{A}$ that runs in time at most T and that does not trigger any of the three events above, we have

$$\Pr[\mathcal{A} \text{ wins}] \leq \frac{1}{2} + \frac{q_s}{2|\text{Dict}|} + \epsilon,$$

where q_s be the maximum number of Send instances that $\mathcal{A}$ initiates.

2.2 UC-Security Definition for PAKE

The UC functionality for PAKE, $\mathcal{F}_{\text{PAKE}}$, is described in Fig. 1.

We define a general version of concrete security in the UC framework as follows.

Definition 2. For a protocol Π and a UC functionality $\mathcal{F}$, we say that Π (T_S, T, ϵ) -realizes $\mathcal{F}$ if there exists a simulator $\mathcal{S}$ that runs in time at most T_S , such that for any environment $\mathcal{Z}$ that runs in time at most T , we have

$$|\Pr[\text{EXEC}_{\Pi, \mathcal{Z}, \mathcal{A}} = 1] - \Pr[\text{EXEC}_{\mathcal{S}, \mathcal{F}, \mathcal{Z}} = 1]| \leq \epsilon.$$

- On $(\text{NewSession}, \text{sid}, P, P', \text{pw})$ from P , send $(\text{NewSession}, \text{sid}, P, P')$ to $\mathcal{A}^*$. If this is the first **NewSession** query for sid , or if this is the second **NewSession** query for sid and there is a record $\langle P', P, \star \rangle$, then record $\langle P, P', \text{pw} \rangle$ and mark it **fresh**.
 - On $(\text{TestPwd}, \text{sid}, P, \text{pw}')$ from $\mathcal{A}^*$, if there is a record $\langle P, P', \text{pw} \rangle$ that is **fresh**, then do:
 - If $\text{pw}' = \text{pw}$, mark the record **compromised** and send “correct guess” to $\mathcal{A}^*$.
 - If $\text{pw}' \neq \text{pw}$, mark the record **interrupted** and send “wrong guess” to $\mathcal{A}^*$.
 - On $(\text{NewKey}, \text{sid}, P, K^*)$ from $\mathcal{A}^*$, where $|K^*| = \lambda$, if there is a record $\langle P, P', \text{pw} \rangle$, and this is the first **NewKey** query for sid and P , then:
 - If the record is **compromised**, send (sid, K^*) to P .
 - If the record is **fresh**, and a key K' was sent to P' , at which time there was a record $\langle P', P, \text{pw} \rangle$ marked **fresh**, send (sid, K') to P .
 - Otherwise, sample $K \leftarrow \{0, 1\}^\lambda$ and send (sid, K) to P .
- Finally, mark the record **completed**.

Figure 1: Functionality $\mathcal{F}_{\text{PAKE}}$

Here, $\mathcal{A}$ is the “dummy” adversary that merely passes all messages between $\mathcal{Z}$ and other parties; $\text{EXEC}_{\Pi, \mathcal{Z}, \mathcal{A}}$ is $\mathcal{Z}$ ’s output bit in the real-world execution of Π with $\mathcal{A}$; and $\text{EXEC}_{\mathcal{S}, \mathcal{F}, \mathcal{Z}}$ is $\mathcal{Z}$ ’s output bit in the ideal-world simulation with $\mathcal{S}$ and $\mathcal{F}$.

There is no dictionary Dict in the UC definition for PAKE, but there has to be an implicit space of all valid values of pw . Below we still use Dict to denote this set, and assume that $|\text{Dict}| \geq 2$ and sampling an element from Dict can be done in time $\mathcal{O}(\lambda)$.³

2.3 Correctness for PAKE

At a high level, correctness means that if there is no active online attack, and the two parties hold the same password, then they output the same session key with overwhelming probability. We present the correctness definition in the UC setting:

Definition 3. For a UC-secure PAKE protocol Π , we say it has correctness error $\epsilon_{\text{correctness}}$, if P and P' have inputs $(\text{NewSession}, \text{sid}, P, P', \text{pw})$ and $(\text{NewSession},$

³For example, [6, Lemma A.1] implicitly defines $\text{Dict} = \{0, 1\}^\lambda$, since the environment in the proof samples $\text{pw} \leftarrow \{0, 1\}^\lambda$.

sid, P', P, pw) respectively for $\text{pw} \leftarrow \text{Dict}$, and the environment instructs the adversary to be passive, then

$$\Pr[P \text{ outputs } (sid, K) \wedge P' \text{ outputs } (sid, K') \wedge K = K' \in \{0, 1\}^\lambda] \geq 1 - \epsilon_{\text{correctness}},$$

where the probability is taken over the random coins used in the protocol execution and the random choice of pw .

Note that we do not require correctness for all possible passwords, as in the environments we use in our proofs, all passwords are sampled uniformly random from the dictionary Dict .

We only present the correctness definition in the UC setting, since our proof relies solely on UC correctness and does not require a separate correctness notion for BPR-security.

2.4 Time Cost Model

As mentioned above, we assume that sampling uniformly at random from the password dictionary Dict can be done in time $\mathcal{O}(\lambda)$.

We assume that basic operations on strings/integers of bitlength ℓ , such as comparing and adding, cost time $\mathcal{O}(\ell)$. Sampling an integer uniformly at random from $[q]$ costs time $\mathcal{O}(\log q)$.

Our environments in the proofs frequently forward messages between different parties (without any modification). To this end, we assume that the time required for forwarding a messages of length at most ℓ (within a single protocol session) is $\mathcal{O}(\ell)$. Let L denote the maximum length of any message sent or received in the protocol. In particular, since initiating a protocol session requires the environment to send a session-initiation message containing the session identifier sid (e.g., $(\text{NewSession}, sid, P, P', \text{pw})$) in PAKE, we have $L > |sid|$ under any standard encoding. For a single session, the total time required for passing messages is $\mathcal{O}(L)$. Note that the exact time depends on the number of message flows in the protocol. In most PAKE protocols, the number of message flows is a small constant, but we keep it as a variable to account for protocols with more, or even polynomially many, message flows.

For general control flow operations, we assume the time consumption of executing a control flow operation is $\mathcal{O}(1)$.

All of the above operations can be done in time $\text{poly}(\lambda)$, if we assume ℓ , L and q are polynomial in λ .⁴

⁴Many concrete security papers simply assume basic operations such as forwarding messages (e.g., between an adversary and a reduction) can be done in time $\text{poly}(\lambda)$, $\mathcal{O}(\lambda)$, or outright ignore them. Strictly speaking this is problematic, since there is no guarantee that the messages sent and read by parties have polynomial length (we stress that we are not in the asymptotic setting).

3 The Main Result

3.1 A Technical Lemma

Before stating our main result, we need a technical lemma that rules out a specific “bad event”. The asymptotic version of the lemma appeared in [6, Appendix A], and a formal proof was given in [10, Appendix A]; here we give a concrete-security version.

Let **SpuriousGuess** be the event that there exists a session id sid such that $\mathcal{Z}$ instructs the adversary to be passive in the session sid , but the simulator $\mathcal{S}$ sends $(\text{TestPwd}, sid, \star, \star)$ to $\mathcal{F}_{\text{PAKE}}$.

Lemma 1. *Suppose Π is a PAKE protocol with dictionary Dict that has correctness error $\epsilon_{\text{correctness}}$ and (T_S, T, ϵ) -realizes $\mathcal{F}_{\text{PAKE}}$. Then for any environment $\mathcal{Z}$ that runs in time at most $T - q \cdot \mathcal{O}(L) - \mathcal{O}(\lambda)$,*

$$\Pr[\text{SpuriousGuess}] \leq \frac{q}{1 - \frac{1}{|\text{Dict}|} - \frac{1}{2^\lambda}} \cdot (\epsilon + \epsilon_{\text{correctness}}),$$

where q is the maximum number of sessions that $\mathcal{Z}$ initiates, and L is the maximum length of any message sent or received in the protocol.

Since we require $|\text{Dict}| \geq 2$, the above can be simplified to

$$\Pr[\text{SpuriousGuess}] \leq 3q(\epsilon + \epsilon_{\text{correctness}}).$$

(In practice when $|\text{Dict}|$ is reasonably large, the bound is very close to $q(\epsilon + \epsilon_{\text{correctness}})$.)

Proof. We construct another environment $\mathcal{Z}_{sp}$ that runs $\mathcal{Z}$ as follows:

1. Sample $i \leftarrow [q]$ as a guess of the index of the session in which **SpuriousGuess** occurs. Sample a random password $\text{pw} \leftarrow \text{Dict}$. Set up a counter $\text{ctr} := 0$ with length $\log q$.
 - Sampling i and setting up the counter take time $\mathcal{O}(\log q)$; sampling pw takes time $\mathcal{O}(\lambda)$. In total, the time consumption is $\mathcal{O}(\log q) + \mathcal{O}(\lambda)$.
2. Run $\mathcal{Z}$ and for each session sid that $\mathcal{Z}$ initiates, increment the counter $\text{ctr} := \text{ctr} + 1$. If $\text{ctr} \neq i$, then forward messages between $\mathcal{Z}$ and P, P' .
 - The operations on the counter can be done in time $\mathcal{O}(\log q)$. Forwarding messages takes time $\mathcal{O}(L)$. There are at most $q - 1$ sessions of this type, so the total time consumption is at most $(q - 1) \cdot \mathcal{O}(L) + \mathcal{O}(\log q)$.
3. If $\text{ctr} = i$, ignore $\mathcal{Z}$, pick any (unique) sid , input $(\text{NewSession}, sid, P, P', \text{pw})$ to P and $(\text{NewSession}, sid, P', P, \text{pw})$ to P' , and instruct the adversary to be passive in the session sid .

- The operations can be done in time $\mathcal{O}(\log q)$ for comparison/control flow and $\mathcal{O}(1)$ for instructing the parties. The time for forwarding the messages is $\mathcal{O}(L)$. In total, the time consumption is $\mathcal{O}(L) + \mathcal{O}(\log q)$.
- 4. After P and P' output the session keys K and K' respectively in the i -th session, output 1 if $K = K'$ and 0 otherwise. If P or P' does not output a session key, output 0.
- The operations can be done in time $\mathcal{O}(\lambda)$ for comparing K and K' .

Overall, the runtime overhead of $\mathcal{Z}_{sp}$ is $q \cdot \mathcal{O}(L) + \mathcal{O}(\lambda)$ (the $\mathcal{O}(\log q)$ term is absorbed into $q \cdot \mathcal{O}(L)$).

Real world analysis. In the real world, since the adversary $\mathcal{A}$ is passive in the i -th session, and the two protocol parties P and P' share the same password, correctness guarantees that

$$\Pr[\text{EXEC}_{\Pi, \mathcal{Z}_{sp}, \mathcal{A}} = 1] = 1 - \epsilon_{\text{correctness}}.$$

Ideal world analysis. Now consider the ideal world. Suppose that the event **SpuriousGuess** happens in the i -th session (i.e., $\mathcal{Z}_{sp}$'s guess is correct). That is, in the i -th session, the simulator $\mathcal{S}$ sends a command $(\text{TestPwd}, \text{sid}, P, \text{pw}^*)$ or $(\text{TestPwd}, \text{sid}, P', \text{pw}^*)$ to $\mathcal{F}_{\text{PAKE}}$, while $\mathcal{Z}$ instructs the adversary to be passive. Other than the i -th session, $\mathcal{Z}_{sp}$ merely forwards messages between $\mathcal{Z}$ and the parties; therefore, the simulator $\mathcal{S}$ sees the same messages under $\mathcal{Z}_{sp}$ as it does under $\mathcal{Z}$. During the i -th session, $\mathcal{Z}_{sp}$ initiates the session and instructs the adversary to eavesdrop just like $\mathcal{Z}$ does, so the simulator $\mathcal{S}$ sees the same messages under $\mathcal{Z}_{sp}$ as it does under $\mathcal{Z}$ as well. Therefore, $\mathcal{S}$'s views — and thus $\mathcal{S}$'s behaviors — under $\mathcal{Z}$ and under $\mathcal{Z}_{sp}$ are exactly identical.

However, $\mathcal{Z}_{sp}$ initiates the i -th session with a password pw that is sampled uniformly at random from Dict independent of everything else (in particular, independent of $\mathcal{S}$'s view). Thus, the probability that $\mathcal{S}$ sends $(\text{TestPwd}, \text{sid}, P, \text{pw}^*)$ or $(\text{TestPwd}, \text{sid}, P', \text{pw}^*)$ with $\text{pw}^* = \text{pw}$ is at most $1/|\text{Dict}|$. Conditioned on the event that $\text{pw}^* \neq \text{pw}$, when P and P' output their session keys K and K' respectively, the probability that $K = K'$ is at most $1/2^\lambda$ as the functionality $\mathcal{F}_{\text{PAKE}}$ outputs a uniformly random session key in $\{0, 1\}^\lambda$ if the instance is interrupted. Thus, we have

$$\Pr[K = K' \mid \text{SpuriousGuess happens in the } i\text{-th session}] = \frac{1}{|\text{Dict}|} + \frac{1}{2^\lambda},$$

so

$$\begin{aligned}
\Pr[\text{EXEC}_{\mathcal{S}, \mathcal{F}_{\text{PAKE}}, \mathcal{Z}_{sp}} = 1] &= \Pr[K = K'] \\
&\leq \Pr[K = K' \mid \text{SpuriousGuess happens in } i\text{-th session}] \\
&\quad \cdot \Pr[\text{SpuriousGuess happens in } i\text{-th session}] \\
&\quad + (1 - \Pr[\text{SpuriousGuess happens in } i\text{-th session}]) \\
&\leq 1 - \left(1 - \frac{1}{|\text{Dict}|} - \frac{1}{2^\lambda}\right) \cdot \Pr[\text{SpuriousGuess happens in } i\text{-th session}] \\
&\leq 1 - \left(1 - \frac{1}{|\text{Dict}|} - \frac{1}{2^\lambda}\right) \cdot \frac{\Pr[\text{SpuriousGuess}]}{q}.
\end{aligned}$$

Putting it together. From the above, the distinguishing advantage of the environment $\mathcal{Z}_{sp}$ is

$$\begin{aligned}
&|\Pr[\text{EXEC}_{\Pi, \mathcal{Z}_{sp}} = 1] - \Pr[\text{EXEC}_{\mathcal{S}, \mathcal{F}_{\text{PAKE}}, \mathcal{Z}_{sp}} = 1]| \\
&\geq \left(1 - \frac{1}{|\text{Dict}|} - \frac{1}{2^\lambda}\right) \cdot \frac{\Pr[\text{SpuriousGuess}]}{q} - \epsilon_{\text{correctness}}.
\end{aligned}$$

Notice that if $\mathcal{Z}$ runs in time at most $T - q \cdot \mathcal{O}(L) - \mathcal{O}(\lambda)$, then $\mathcal{Z}_{sp}$ runs in time at most T . Since the protocol Π (T_S, T, ϵ) -realizes $\mathcal{F}_{\text{PAKE}}$, we have that the distinguishing advantage of $\mathcal{Z}_{sp}$ is at most ϵ . Thus, we have

$$\left(1 - \frac{1}{|\text{Dict}|} - \frac{1}{2^\lambda}\right) \cdot \frac{\Pr[\text{SpuriousGuess}]}{q} - \epsilon_{\text{correctness}} \leq \epsilon,$$

which implies that

$$\Pr[\text{SpuriousGuess}] \leq \frac{q}{1 - \frac{1}{|\text{Dict}|} - \frac{1}{2^\lambda}} \cdot (\epsilon + \epsilon_{\text{correctness}}). \quad \square$$

3.2 Concrete Security from UC PAKE to BPR

Let q_e (resp. q_s) be the number of Execute (resp. Send) commands that the game-based adversary makes. Our main result is:

Theorem 2. *Suppose Π is a PAKE protocol with dictionary Dict that has correctness error $\epsilon_{\text{correctness}}$ and (T_S, T, ϵ) -realizes $\mathcal{F}_{\text{PAKE}}$. Then Π is (T', ϵ') -BPR-secure, where*

$$\begin{aligned}
T' &= T - (3q + q_s) \cdot \mathcal{O}(L) - 2q \cdot \mathcal{O}(\lambda), \\
\epsilon' &= \epsilon + \frac{q}{1 - \frac{1}{|\text{Dict}|} - \frac{1}{2^\lambda}} \cdot (\epsilon + \epsilon_{\text{correctness}}).
\end{aligned}$$

Here, q_s is the Send instances that the BPR adversary initiates, q is the total number of instances, and L is the maximum length of any message sent or received in the protocol.

The bound for ϵ' can be simplified to

$$\epsilon' \leq \epsilon + 3q(\epsilon + \epsilon_{\text{correctness}}) = (3q + 1)\epsilon + 3q\epsilon_{\text{correctness}}.$$

The proof was first done in [6, Appendix A]; our presentation follows [11, Section 4] more closely.

Proof. Let $\hat{\mathcal{A}}$ be a BPR adversary against Π that runs in time T' . Recall that $\hat{\mathcal{A}}$ initiates q instances in total and q_s **Send** instances; let $q_e = q - q_s$ be the number of **Execute** instances that $\hat{\mathcal{A}}$ initiates. We construct an environment $\mathcal{Z}_{gb}$ that runs $\hat{\mathcal{A}}$ as follows: (below we use $\mathcal{A}$ to denote the dummy UC adversary controlled by $\mathcal{Z}_{gb}$)

1. On **Execute**($P^i, (P')^j$) from $\hat{\mathcal{A}}$, sample and store a fresh $sid_{i,j} = sid_{j,i}$, sample a password $\text{pw} \leftarrow \text{Dict}$, and send (**NewSession**, $sid_{i,j}, P, P', \text{pw}$) to P and (**NewSession**, $sid_{i,j}, P', P, \text{pw}$) to P' . After that, instruct $\mathcal{A}$ to be passive in this session. When the session completes, forward the transcript to $\hat{\mathcal{A}}$.
 - The cost of storing and retrieving $sid_{i,j}$ is upper-bounded by $\mathcal{O}(L)$, since $L > |sid_{i,j}|$ and the index (i, j) enables efficient access to the corresponding entry. Sampling pw takes time $\mathcal{O}(\lambda)$; passively forwarding messages takes time $\mathcal{O}(L)$. In total, the time consumption is $\mathcal{O}(L) + \mathcal{O}(\lambda)$ for one session and $q_e \cdot (\mathcal{O}(L) + \mathcal{O}(\lambda))$ for all sessions.
2. On **Send**($P^i, (P')^j, \text{start}$) from $\hat{\mathcal{A}}$, check if $sid_{i,j} = sid_{j,i}$ has been sampled. If so, retrieve pw corresponding to (i, j) and send (**Send**, $sid_{i,j}, P^i, (P')^j, \text{pw}$) to P . Otherwise, sample a fresh $sid_{i,j} = sid_{j,i}$, sample and store a password $\text{pw} \leftarrow \text{Dict}$ with indices (i, j) , and send (**NewSession**, $sid_{i,j}, P, P', \text{pw}$) to P . Forward the message $\mathcal{A}$ receives to $\hat{\mathcal{A}}$.
 - The time consumption is $\mathcal{O}(L) + \mathcal{O}(\lambda)$ for one session and $q_s \cdot (\mathcal{O}(L) + \mathcal{O}(\lambda))$ for all sessions.
3. On **Send**($P^i, (P')^j, m^*$) from $\hat{\mathcal{A}}$, instruct $\mathcal{A}$ to send m^* to P' . Forward the message $\mathcal{A}$ receives to $\hat{\mathcal{A}}$.
 - The time consumption is $\mathcal{O}(L)$ for one session and $q_s \cdot \mathcal{O}(L)$ for all sessions.
4. On **Reveal**(P^i) from $\hat{\mathcal{A}}$, retrieve the session that P^i and $(P')^j$ communicates, $sid_{i,j}$. If P has output $(sid_{i,j}, K)$, send K to $\hat{\mathcal{A}}$; otherwise, send $\perp$.
 - The time is $\mathcal{O}(L)$ for retrieving $sid_{i,j}$ and $\mathcal{O}(\lambda)$ for forwarding K (or $\perp$). Overall, the time consumption is $\mathcal{O}(L) + \mathcal{O}(\lambda)$ for one session and $q \cdot (\mathcal{O}(L) + \mathcal{O}(\lambda))$ for all sessions.

5. On $\text{Test}(P^i)$ from $\hat{\mathcal{A}}$, sample a bit $b \leftarrow \{0, 1\}$. Retrieve the session that P^i and $(P')^j$ communicates, $\text{sid}_{i,j}$. If P has returned $(\text{sid}_{i,j}, K)$ and $b = 1$, send K to $\hat{\mathcal{A}}$; if P has returned $(\text{sid}_{i,j}, K)$ and $b = 0$, sample a random string in $\{0, 1\}^\lambda$ and send it to $\hat{\mathcal{A}}$; if P has not returned anything, send $\perp$ to $\hat{\mathcal{A}}$.
 - The time consumption is the same as the **Reveal** case, except some control flow and sampling. Thus, the time consumption is $\mathcal{O}(L) + \mathcal{O}(\lambda)$ (note that there can be only one **Test** command).
6. When $\hat{\mathcal{A}}$ outputs a bit b' , output 1 if $b = b'$ and 0 otherwise.
 - The operations can be done in time $\mathcal{O}(1)$.

Overall, the runtime overhead of $\mathcal{Z}_{gb}$ is $(2q + q_s) \cdot \mathcal{O}(L) + 2q \cdot \mathcal{O}(\lambda)$. It is clear that the number of UC sessions initiated by $\mathcal{Z}_{gb}$ does not exceed q .⁵

Real world analysis. In the real world, it is clear that

$$\Pr[\text{EXEC}_{\Pi, \mathcal{Z}_{gb}, \mathcal{D}} = 1] = \Pr[\hat{\mathcal{A}} \text{ wins}].$$

Ideal world analysis. Let $\mathcal{S}$ be a simulator running in time at most T_S , such that for any environment $\mathcal{Z}$ that runs in time at most T , $\mathcal{S}$ generates a view whose distinguishing advantage from $\mathcal{Z}$'s real-world view is at most ϵ . Define **CorrectGuess** as the event that the simulator sends $(\text{TestPwd}, \text{sid}, P, \text{pw})$ to $\mathcal{F}_{\text{PAKE}}$ resulting in “correct guess”.

Suppose **SpuriousGuess** does not happen. Then $\mathcal{S}$ sends at most q_s **TestPwd** commands to $\mathcal{F}_{\text{PAKE}}$. Since each password is sampled uniformly from the dictionary, independent of everything else, the probability of obtaining “correct guess” is at most $q_s/|\text{Dict}|$. Thus, we have

$$\Pr[\text{CorrectGuess} \mid \overline{\text{SpuriousGuess}}] \leq \frac{q_s}{|\text{Dict}|}.$$

Next, suppose **CorrectGuess** does not happen. Let P^i be the instance for which $\hat{\mathcal{A}}$ sends **Test**. By step 5 of $\mathcal{Z}_{gb}$, a session id $\text{sid}_{i,j}$ is associated with P^i . We have the following two cases about $\text{sid}_{i,j}$:

- If no $(\text{TestPwd}, \text{sid}_{i,j}, P, \star)$ command is sent to $\mathcal{F}_{\text{PAKE}}$, P 's session record is marked **fresh**, so $\mathcal{F}_{\text{PAKE}}$ samples a uniformly random key for P independent of $\hat{\mathcal{A}}$'s view. (If P' 's session record is also **fresh**, P and P' 's session keys are equal. However, in this case $\hat{\mathcal{A}}$ is not allowed to **Reveal** P' according to the prohibited behaviors in Section 2.1, so P 's session key is independent of $\hat{\mathcal{A}}$'s view.)

⁵Recall that q is the number of protocol instances generated by $\hat{\mathcal{A}}$, and by the definition of $\mathcal{Z}_{gb}$, it may initiate a session only when $\hat{\mathcal{A}}$ generates an instance. Since each session contains one or two instances, $\mathcal{Z}_{gb}$ might initiate fewer than q sessions, and minor optimizations on the bound might be possible (which we omit here).

- If a $(\text{TestPwd}, \text{sid}_{i,j}, P, \star)$ command is sent to $\mathcal{F}_{\text{PAKE}}$, since we assume that CorrectGuess does not happen, P 's session record is **interrupted**, so $\mathcal{F}_{\text{PAKE}}$ also samples a uniformly random key for P independent of $\hat{\mathcal{A}}$'s view.

Either way, P 's session key (and thus the bit b) is independent of $\hat{\mathcal{A}}$'s view. Thus,

$$\Pr[b^* = b \mid \overline{\text{CorrectGuess}}] = \frac{1}{2}.$$

The runtime of $\mathcal{Z}_{gb}$ is up to $T' + (2q + q_s) \cdot \mathcal{O}(L) + 2q \cdot \mathcal{O}(\lambda) \leq T - q \cdot \mathcal{O}(L) - \mathcal{O}(\lambda)$. Since $\mathcal{Z}_{gb}$ initiates at most q sessions, by Lemma 1, we have

$$\Pr[\text{SpuriousGuess}] \leq \frac{q}{1 - \frac{1}{|\text{Dict}|} - \frac{1}{2^\lambda}} \cdot (\epsilon + \epsilon_{\text{correctness}}).$$

Summing up, we get

$$\begin{aligned} \Pr[\text{EXEC}_{S, \mathcal{F}_{\text{PAKE}}, \mathcal{Z}_{gb}, \text{Dict}} = 1] &\leq \Pr[\text{SpuriousGuess}] + \Pr[b^* = b \mid \overline{\text{SpuriousGuess}}] \\ &\leq \Pr[\text{SpuriousGuess}] + \Pr[\overline{\text{CorrectGuess}} \mid \overline{\text{SpuriousGuess}}] \\ &\quad + \Pr[b^* = b \wedge \overline{\text{CorrectGuess}} \mid \overline{\text{SpuriousGuess}}] \\ &\leq \Pr[\text{SpuriousGuess}] + \Pr[\overline{\text{CorrectGuess}} \mid \overline{\text{SpuriousGuess}}] \\ &\quad + \frac{1}{2}(1 - \Pr[\overline{\text{CorrectGuess}} \mid \overline{\text{SpuriousGuess}}]) \\ &\leq \frac{1}{2} + \frac{1}{2} \Pr[\overline{\text{CorrectGuess}} \mid \overline{\text{SpuriousGuess}}] + \Pr[\text{SpuriousGuess}] \\ &\leq \frac{1}{2} + \frac{q_s}{2|\text{Dict}|} + \frac{q}{1 - \frac{1}{|\text{Dict}|} - \frac{1}{2^\lambda}} \cdot (\epsilon + \epsilon_{\text{correctness}}). \end{aligned}$$

Putting it together. Comparing the environment $\mathcal{Z}_{gb}$'s outputs in the real world and in the ideal world, we conclude that $\mathcal{Z}_{gb}$'s distinguishing advantage is at least

$$\Pr[\hat{\mathcal{A}} \text{ wins}] - \frac{1}{2} - \frac{q_s}{2|\text{Dict}|} - \frac{q}{1 - \frac{1}{|\text{Dict}|} - \frac{1}{2^\lambda}} \cdot (\epsilon + \epsilon_{\text{correctness}}).$$

Since Π is (T_S, T, ϵ) -secure, we have

$$\Pr[\hat{\mathcal{A}} \text{ wins}] - \frac{1}{2} - \frac{q_s}{2|\text{Dict}|} - \frac{q}{1 - \frac{1}{|\text{Dict}|} - \frac{1}{2^\lambda}} \cdot (\epsilon + \epsilon_{\text{correctness}}) \leq \epsilon,$$

so

$$\begin{aligned} \Pr[\hat{\mathcal{A}} \text{ wins}] &\leq \frac{1}{2} + \frac{q_s}{2|\text{Dict}|} + \epsilon + \frac{q}{1 - \frac{1}{|\text{Dict}|} - \frac{1}{2^\lambda}} \cdot (\epsilon + \epsilon_{\text{correctness}}) \\ &= \frac{1}{2} + \frac{q_s}{2|\text{Dict}|} + \epsilon', \end{aligned}$$

which completes the proof. $\square$

4 Discussion

As mentioned in Section 1, our work makes little technical contribution: its essence is to “open up” the proof of [6, Section A] and carefully calculate the exact parameters. Rather, the main contribution is a conceptual one, namely bridging the gap between the UC and game-based security notions for PAKE. In fact, there has been much discussion on “UC vs. game-based” in recent years; below we only discuss one work that touches upon this topic.

[9] is titled “EKE Meets Tight Security in the Universally Composable Framework”. Upon seeing the title, one may immediately wonder what the term “tight” means in UC: if, say, the reduction is tight when the simulator runs in time $\mathcal{O}(\lambda^3)$ but non-tight when the simulator runs in time $\mathcal{O}(\lambda)$, then does it count as “tight” or not?

This question is not explicitly answered in [9]. Indeed, its main result (Theorem 2) gives concrete bounds for the environment’s advantage, but only requires the simulator to be “PPT”. As such, it seems that the theorem is partly in the concrete security setting and partly in the asymptotic setting. However, our result shows that the simulator’s runtime does not matter, so “tightness” for UC PAKE is in fact well-defined: only the environment’s runtime T and its advantage ϵ need to be taken into account, and tightness can be defined similar to the game-based setting. (This, of course, assumes that their reductions’ runtime is in the “normal” range — cf. the discussion at the end of Section 1 — which is indeed the case upon inspection.)

Future directions. It appears that other types of “UC implies game-based” statements can be made concrete as well. [1] suggests that whenever a new UC functionality is proposed, we should perform a “sanity check” that UC-security implies some more intuitive game-based security notion, in order to ensure that our UC functionality is a reasonable one. This work gives another reason for doing this check: *it helps interpret UC-security more easily in the concrete security setting, by converting it into a game-based security statement first*. In other words, with a “UC implies game-based” statement, a critical drawback of UC diminishes.

Finally, we conjecture that in a large number of such statements, the simulator’s runtime should be irrelevant — similar to our Theorem 2. The rationale is that in the proof of such a statement, we usually “translate” the game-based attacking advantage into the ideal-world attacking advantage, which is generally information-theoretical. If this intuition turns out to be true, then the concept of “pseudo-UC-security” — where the simulator is computationally unbounded — may find further applications (despite that composability does not hold anymore).

References

- [1] Myrto Arapinis, Véronique Cortier, Hubert de Groote, Charlie Jacomme, and Steve Kremer. Are ideal functionalities really ideal? Cryptology ePrint Archive, Report 2025/2125, 2025.
- [2] Jiawei Bao, Tibor Jager, Eike Kiltz, Aysan Nishaburi, Samin Nooripoor, and Jiaxin Pan. Beyond 128 bits: The concrete security of EKE. Cryptology ePrint Archive, Report 2026/1053, 2026.
- [3] Mihir Bellare, David Pointcheval, and Phillip Rogaway. Authenticated key exchange secure against dictionary attacks. In Bart Preneel, editor, *EUROCRYPT 2000*, volume 1807 of *LNCS*, pages 139–155. Springer, Berlin, Heidelberg, May 2000.
- [4] Mihir Bellare, Rishabh Ranjan, Doreen Riepel, and Ali Aldakheel. The concrete security of two-party computation: Simple definitions, and tight proofs for PSI and OPRFs. In Kai-Min Chung and Yu Sasaki, editors, *ASIACRYPT 2024, Part VI*, volume 15489 of *LNCS*, pages 328–362. Springer, Singapore, December 2024.
- [5] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd FOCS*, pages 136–145. IEEE Computer Society Press, October 2001.
- [6] Ran Canetti, Shai Halevi, Jonathan Katz, Yehuda Lindell, and Philip D. MacKenzie. Universally composable password-based key exchange. In Ronald Cramer, editor, *EUROCRYPT 2005*, volume 3494 of *LNCS*, pages 404–421. Springer, Berlin, Heidelberg, May 2005.
- [7] Crypto Forum Research Group. PAKE selection, 2020. <https://github.com/cfrg/pake-selection>.
- [8] Naman Kumar and Jiayu Xu. On the UC-(in)security of PAKE protocols without the random oracle model. Cryptology ePrint Archive, Report 2025/998, 2025.
- [9] Xiangyu Liu, Shengli Liu, Shuai Han, and Dawu Gu. EKE meets tight security in the Universally Composable framework. In Alexandra Boldyreva and Vladimir Kolesnikov, editors, *PKC 2023, Part I*, volume 13940 of *LNCS*, pages 685–713. Springer, Cham, May 2023.
- [10] Lawrence Roy and Jiayu Xu. A universally composable PAKE with zero communication cost (and why it shouldn’t be considered UC-secure). In Alexandra Boldyreva and Vladimir Kolesnikov, editors, *PKC 2023, Part I*, volume 13940 of *LNCS*, pages 714–743. Springer, Cham, May 2023.
- [11] Jiayu Xu. Does the UC-security notion for PAKE imply game-based security? *CiC*, 2(4):11, 2026.

- [12] Jiayu Xu and Zhiyuan Zhao. Just how secure is SRP, really? Cryptology ePrint Archive, Report 2025/1869, 2025.