UC-Security of Encrypted Key Exchange: A Tutorial *

Jiayu Xu

Oregon State University
xujiay@oregonstate.edu

Abstract

Password-Authenticated Key Exchange (PAKE) is a type of key exchange protocols secure against man-in-the-middle adversaries, in the setting where the two parties only agree upon a low-entropy "password" in advance. The first and arguably most well-studied PAKE protocol is Encrypted Key Exchange (EKE) (Bellovin and Marritt, 1992), and the standard security notion for PAKE is in the Universal Composability (UC) framework (Canetti et al., 2005). While the UC-security of EKE has been "folklore" knowledge for many years, a satisfactory formal proof has long been elusive.

In this work, we present a UC-security proof for the most common instantiation of EKE, which is based on hashed Diffie–Hellman. Our proof is in the random oracle + ideal cipher models, and under the computational Diffie–Hellman assumption. We thoroughly discuss the UC-security definition for PAKE, subtleties and pitfalls in the security proof, how to write a UC proof, and flaws in existing works; along the way we also present some philosophical discussions on security definitions and security proofs in general. In this way, we hope to draw attention to several understudied, underexplained or underappreciated aspects of the UC-security of EKE.

This tutorial can be viewed as a simplified version of the recent work by Januzelli, Roy and Xu (2025); however, we completely rewrite most of the materials there to make them much more approachable to beginners who have just learned the UC framework.

*This is a handout for my CS 599 (Topics in Cryptography) course at Oregon State University, taught in winter 2025. We went over Sections 2 and 3 in four 80-minute lectures. I thank all students in my class for helpful (and fun!) discussions.

Contents

1 Introduction

> Student: The car has a speed of 50 miles an hour. What does that mean?
> Teacher: Given any $\varepsilon > 0$, there exists a δ such that if $[0 <]|t_2 - t_1| < \delta$, then
>
> $$\left|\frac{s_2 - s_1}{t_2 - t_1} - 50\right| < \varepsilon.$$
>
> Student: How in the world did anybody ever think of such an answer?
>
> Judith V. Grabiner [Gra83]

Imagine you are a first-year PhD student who has just started working in cryptography. Your advisor suggested that you look into *Password-Authenticated Key Exchange (PAKE)*, a type of two-party protocols of both theoretical and practical significance. You learned that PAKE can be viewed as an extension of basic Key Exchange (KE) protocols such as Diffie–Hellman, but secure against man-in-the-middle adversaries. Of course, in the man-in-the-middle setting there needs to be some trusted setup; otherwise nothing prevents an adversary from impersonating one party and learning the other party's key. The setup for PAKE is called *password-only*, where the two parties only agree upon a low-entropy string called the password in advance, meaning that the set of all possible passwords (the "dictionary") has polynomial size; a PAKE protocol essentially bootstraps this weak string to a cryptographically strong key. At an intuitive level, everything makes sense.

Naturally, the next step was to see the formal security definition for PAKE. You found out that there were two such definitions in the literature: *game-based* [BPR00] and *Universally Composable (UC)* [CHK+05]. The UC definition came 5 years later, but has gradually been accepted as the "standard" one, since it models arbitrary composition, with PAKE itself or with whatever other protocol on top of it — which is the common application of PAKE, where parties use the output key to further perform some symmetric cryptographic operations.

This again makes sense, so you started actually reading the security definitions. In both definitions, the key idea is to make sure that *the adversary can only perform a single online guessing attack per instance* — same as the aforementioned man-in-the-middle attack on Diffie–Hellman KE, but this time the adversary needs to guess the correct password in order to succeed; this happens with non-negligible probability but is inevitable, and we want to make sure that this is the only form of attacks that are feasible. Now things become a little messier: like the (ε, δ)-definition of the limit of a function, the security definitions for PAKE — both game-based and UC — look pretty far from the intuition; in particular, there are some complicated sentences in the UC PAKE functionality about the "state" of an instance and how the instance's output key depends on it, and you are not sure if you fully understand all of the details here.

In order to further grasp the security definition(s), you decided to find a concrete PAKE protocol and read its security proof. There are dozens in the literature, but one name appears again and again: *Encrypted Key Exchange (EKE)*. Introduced in 1992 [BM92], this was the first PAKE protocol that marked the beginning of PAKE as an area of study; furthermore, it has inspired a number of follow-up works, and some of the highest-performance PAKEs nowadays are essentially variants of EKE. This is a very elegant and simple protocol: take a basic KE such as Diffie–Hellman, encrypt the messages under the password using an Ideal Cipher (IC), and send the resulting ciphertexts as

PAKE messages; each party — assuming it holds the correct password — can decrypt and recover the underlying KE messages, and then perform KE operations to derive the session key.

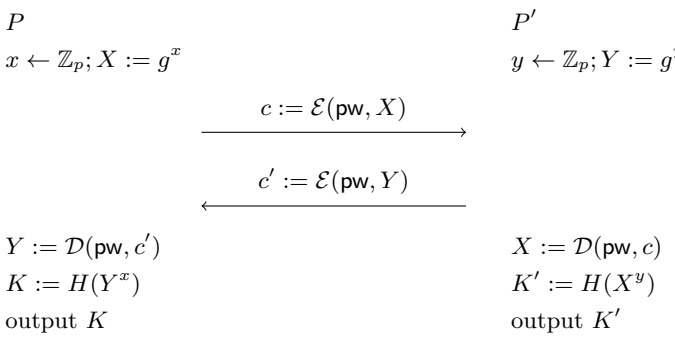


Figure 1: EKE with hashed Diffie–Hellman

The EKE protocol is easy to understand, so you proceeded to look at its security proof. The game-based proof was done in 2000, in the same paper that introduced the game-based definition [BPR00]: this version assumes hashed Diffie–Hellman as the underlying KE, and if the hash is modeled as a Random Oracle (RO), then EKE is secure under the Computational Diffie–Hellman (CDH) assumption. However, when you tried to find a UC proof, you ran into a major obstacle: there didn't seem to be one! How does such a simple and classic protocol not have a standard UC-security proof? And how are you supposed to understand the UC-security definition without reading a proof in UC?

Getting lost, you decided to move on to some other areas in cryptography. A few years have passed, and you have now forgotten much about PAKE. Then all of a sudden, a large number of works proving the UC-security of EKE popped up in the span of several years [DHP+18, MRR20, BCP+23, FGJ23] — perhaps because the IETF has started a standardization process for PAKE [Cry20]. You tried to read them, but this task appears prohibitive, as there are some subtle differences among these results: some assume EKE with hashed Diffie–Hellman just as in the game-based security proof, while others prove the UC-security of EKE with *any* underlying KE (that satisfies certain properties); in particular, some of these results imply that outputting the "raw" Diffie–Hellman key g^{xy} suffices, and there is no need to hash it using an RO. Instead of clarifying the UC PAKE definition, these recent works confused you even further: Is it necessary to hash the Diffie–Hellman key to achieve UC-security? And why is there not a single clearly written security proof for this arguably most studied PAKE protocol under the standard UC definition?

1.1 Comparison with [JRX25]

The above was a brief history of EKE until very recently. The UC-security of EKE was finally resolved in [JRX25], which points out a number of flaws in existing works' security statements for EKE, and gives a proof for a more efficient version of EKE. This tutorial is largely based on [JRX25], but we completely rewrite most of the materials:

- The UC PAKE functionality in Section 2.1, originally from [CHK+05], is presented and explained in [JRX25, Section 2.2]; however, we give a more detailed explanation of its intuitions,

as well as why UC-security has become the standard for PAKE. (Many students find the standard UC PAKE functionality hard to understand, and it is impossible to write a correct UC proof without thoroughly understanding the functionality — see Section 4.)

- Section 2.2 shows that EKE with plain Diffie–Hellman (i.e., outputting the "raw" Diffie–Hellman key) is *not* UC-secure, which appeared first in a talk [Jar23] and then in written form in [JRX25, Section 3.2].
- In Section 2.3 we show why the UC-security of EKE (with hashed Diffie–Hellman) needs an assumption called 1-ODH, and why its reduction to CDH loses a *quadratic* factor. This is discussed in [JRX25, Appendix A.1]; here we stress this point again, which has been overlooked by all other works on the UC-security of EKE.
- Section 3 presents the UC-security proof for EKE, which is essentially a simplified version of [JRX25, Sections 5.1 and 5.2].
- Discussions on the flaws in existing works in Section 4 are scattered throughout [JRX25, Section 3.2 and Appendix A.1]. Some of the opinions at the end are expressed in [JRX25, Section 6], but most of them are my own.
- I came up with some of the exercises in Section 5 myself; other exercise questions were asked by students in my CS 599 class.

Apart from what we cover here, [JRX25] treats EKE in a much more general setting:

- The EKE protocol analyzed in [JRX25] assumes *any* underlying KE protocol that satisfies certain properties, which covers hashed Diffie–Hellman as a special case. Using Diffie–Hellman-type KE significantly simplifies the security analysis: (1) it has perfect correctness; (2) it has perfect pseudorandomness, meaning that the protocol messages g^x, g^y are uniformly random, rather than just pseudorandom, in their space; and (3) it is 1-simultaneous round, i.e., party P''s message g^y does not depend on P's message g^x. Many post-quantum KE protocols satisfy none of these three properties, making the EKE security proof harder.
- [JRX25] uses a randomized version of IC called programmable-once public function, which is roughly 4 times faster than IC. The fact that $\mathcal{E}(\mathsf{pw}, X; r)$ and $\mathcal{E}(\mathsf{pw}, X; r')$ for $r \neq r'$ may result in different ciphertexts further complicates the proof.
- Finally, [JRX25] also proves the UC-security of a variant of EKE called one-encryption EKE.

While [JRX25] aims for maximal generality, it somewhat sacrifices the readability, as there are multiple orthogonal subtleties in the same proof. By using IC and the specific hashed Diffie–Hellman KE, we "filter out" some of the complications and focus on the core of this proof. We drastically improve readability from [JRX25] by completely rewriting most of the materials.

We now highlight some minor features of this tutorial. Many UC proofs, after starting from the real world and showing the necessary hybrids, claim that "by inspection of the simulator" the last hybrid is identical to the ideal world. We believe this is a dangerous approach that might render the proof incorrect; instead, we present such arguments in detail. Also, in a UC proof there are many formalisms that can be ignored or simplified once we make proper declarations (e.g., assuming the real adversary is "dummy", and omitting the session ID in some messages); we show how exactly this should be done right before and at the beginning of our proof.

2 Preliminaries

> All mathematicians are familiar with the concept of an *open research problem*. I propose the less familiar concept of an *open exposition problem*. Solving an open exposition problem means explaining a mathematical subject in a way that renders it totally perspicuous. Every step should be motivated and clear; ideally, students should feel that they could have arrived at the results themselves.
>
> Timothy Y. Chow [Cho09]

Let n be the security parameter. We will use a cyclic group $(\mathbb{G}, g, p)$ where the order p is superpolynomial in n.[1]

2.1 The UC PAKE Functionality

See Figure 2 for the UC functionality for PAKE.

- 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 record $\langle P, P', \mathsf{pw}\rangle$ 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 $(\mathsf{NewKey}, sid, P, K^* \in \{0,1\}^n)$ 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, K) to P, where K is defined as follows:
  - If the record is compromised, then set $K := K^*$.
  - If the record is fresh, a key (sid, K') has been output to P', at which time there was a record $\langle P', P, \mathsf{pw}\rangle$ marked fresh, then set $K := K'$. // if no attack in this session, and passwords match, then keys should also match
  - Otherwise sample $K \leftarrow \{0,1\}^n$.

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

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

[1] To be precise, since we are in the asymptotic setting, we must consider an infinite sequence of groups. That is, there needs to be a group generation algorithm GenGroup which on input 1^n generates $(\mathbb{G}, g, p)$ for p superpolynomial in n, and all group-based assumptions should be w.r.t. GenGroup. Below we abuse notations and simply say that the assumptions hold in $(\mathbb{G}, g, p)$.

This functionality was explained in the original UC PAKE paper [CHK+05, Section 2] and then in [RX23, Section 2.2] and [JRX25, Section 2.2]; here we give a self-contained explanation that is (hopefully) more approachable to beginners. The core of this functionality is to formalize two intuitive principles, which any reasonable security definition for PAKE should satisfy:

The first principle: The adversary can only perform a single online guessing attack per instance. This is modeled via the TestPwd command, where the ideal adversary $\mathcal{A}^*$ uses a password guess pw^* to attack instance (sid, P). Crucially, such a command can be sent only when this instance is fresh, and once a TestPwd is sent, the instance will become compromised or interrupted (depending 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 can be sent per instance.

The second principle: All session keys are independent of each other, except in cases where correlation is inevitable. But what are such cases?

- If the adversary is passive (i.e., passes all protocol messages without modification) and the two parties' passwords match, then correctness of the protocol should guarantee that the two parties' session keys are equal. This is modeled in the second bullet under NewKey: if both P and P''s instances are fresh when they output their (respective) session keys, and P' outputs K' first, then when P outputs, its session key K is equal to K'.

- If an instance has been successfully attacked, then all security guarantees for this instance are lost and we cannot claim any independence here. In the first bullet under NewKey, the functionality simply lets $\mathcal{A}^*$ choose the instance's session key. (Note that in all other cases, the session key K^* specified by $\mathcal{A}^*$ is not actually used.)

- The third bullet under NewKey is an umbrella case that covers a number of sub-cases: (1) the instance is unattacked and outputs before its counterparty's instance does; (2) both instances are unattacked, but their passwords differ; (3) the instance is unsuccessfully attacked; and (4) the instance is unattacked, and its counterparty's instance is (successfully or unsuccessfully) attacked. In all these cases, the functionality lets the instance output a uniformly random session key; in (2)(3)(4), it is guaranteed that this session key is independent of all other session keys. (In (1), if later the counterparty's instance is also unattacked, then the two parties output the same key — as we have seen in the second bullet under NewKey.)

The above explains the TestPwd and NewKey commands. The NewSession command is for protocol parties to start a new instance; in the ideal world, once the simulator receives such a message from the functionality, it should start simulating the first protocol message of this instance.

Why UC? We briefly mentioned in Section 1 that the UC-security definition has superceded the game-based definition and become *the* security definition for PAKE, and one of the reasons is that UC guarantees security even under arbitrary composition. Another often-mentioned reason is that the UC functionality lets the environment — which is an adversarial party — choose the password; this means that it naturally models *arbitrary password distribution*, including e.g., mistyped passwords or highly correlated passwords across multiple accounts. By contrast, in the game-based definition we always assume an a priori fixed distribution of passwords over the dictionary, which is weaker and does not fit real-world applications of passwords well.

We also wish to highlight some more theoretical/conceptual advantages of the UC definition, which are rarely mentioned in the literature. First, it is clear that the two principles above are the

"right" intuition behind PAKE security, but formalizing them turns out to be highly non-trivial; the UC functionality offers an arguably cleaner way to do it, with the first principle clearly modeled by TestPwd and the second clearly modeled by NewKey.

Second, UC integrates multiple security requirements into a single functionality, which the game-based definition fails to achieve. For example, in the game-based definition, the adversary simply "wins" once it guesses the correct password, and there is no guarantee whatsoever on what might happen after that. A common requirement for PAKE is *forward secrecy*, which means that even if the password is leaked, the session keys of instances already completed should remain secure. In the game-based setting, this must be defined separately, so one needs to do two proofs for a PAKE protocol. On the other hand, it is clear from the UC PAKE functionality that forward secrecy is guaranteed there: once an instance outputs its session key, it is marked completed which disallows any further commands from the ideal adversary. (Note that TestPwd can be sent only when the instance is fresh.) This means that the adversary cannot do anything on a completed instance even if it knows the password, which is exactly forward secrecy.[2]

2.2 Motivating Example: EKE with Plain Diffie–Hellman Is Not UC-Secure

In this section, we will first answer the question in the last paragraph before Section 1.1; that is, EKE which outputs the "raw" Diffie–Hellman key g^{xy} is not UC-secure[3], and to achieve UC-security the Diffie–Hellman key has to be hashed. This might be surprising to some: while it has long been "folklore" knowledge that EKE is UC-secure, it seems that people could not agree upon whether the Diffie–Hellman key should be hashed or not, and some have held the false belief that outputting the "raw" key suffices.[4]

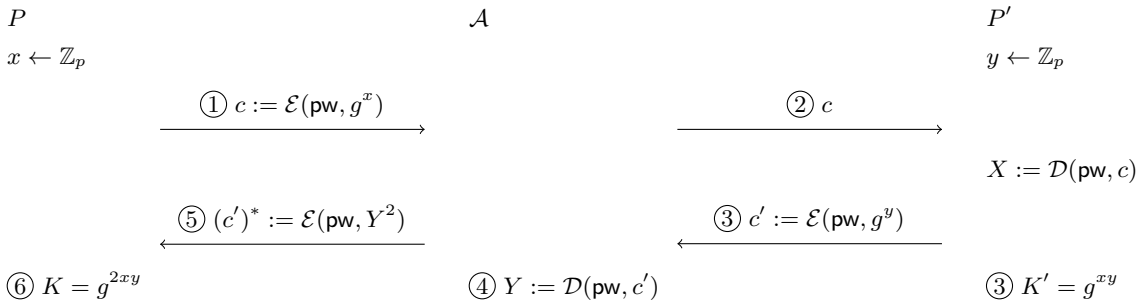


Figure 3: Attack on EKE with plain Diffie–Hellman that renders it UC-inseucre. The adversary passes the first message but modifies the second, causing the two parties' session keys correlated

[2]There are other security requirements incorporated into the UC notion but not the game-based one: (1) the game-based notion does not consider the scenario where the two parties' passwords are different, whereas UC-security requires the two parties's session keys to be independently random in this case; and (2) the game-based notion requires the adversary to decide whether to be passive or active in a session before the session starts, whereas in UC the adversary can adaptively make this decision based on (say) the first two protocol messages it sees.

[3]Here we slightly tweak the UC PAKE functionality so that the output key on NewKey is in $\mathbb{G}$, not $\{0,1\}^n$.

[4]Of course, in practice people generally hash the output key. But the question whether outputting the "raw" key suffices for UC-security is still meaningful: in particular, if a security proof does not give a "yes/no" answer to this question then it is unclear, and if it gives a "yes" answer then it is incorrect.

Consider the attack on EKE in Figure 3, where the man-in-the-middle adversary passes the first message c without modification, but decrypts the second message c' using the correct password, obtains $Y = g^y$, and re-encrypts $Y^2 = g^{2y}$; this causes P's session key K to be the square of P''s session key K'.

Let's attempt a UC simulation of the real adversary above. The simulator $\mathcal{S}$ does not learn any information about pw until steps 4 and 5, when $\mathcal{A}$ decrypts $Y := \mathcal{D}(\mathsf{pw}, c')$ and then re-encrypts $(c')^* := \mathcal{E}(\mathsf{pw}, Y^2)$; crucially, pw is not used anywhere before that. Therefore,

- $\mathcal{S}$ needs to let P' output K' (via a NewKey command) in step 3. Since there has been no attack and $\mathcal{S}$ has no knowledge about pw, it should not send $\mathsf{TestPwd}$ for P'; therefore, P''s session is fresh and K' is a random string independent of everything else.
- Later, when $\mathcal{S}$ learns pw in steps 4 and 5, it is "too late" and $\mathcal{S}$ cannot learn anything about K'; the maximum $\mathcal{S}$ can do is to use a $\mathsf{TestPwd}$ command on pw to compromise P's instance, but it cannot do anything on the P' side. (Recall that the PAKE functionality does not allow the ideal adversary to send any command on a completed instance; see the discussion about forward secrecy at the end of Section 2.1.)
- When P outputs K in step 6, this still does not change the fact that K' is independent of everything else.

However, this is not the case in the real world, where $K = (K')^2$. This suggests an environment that can distinguish between the real world and the ideal world: run P and P' on the same password pw, instruct the adversary to proceed as in Figure 3, and observe the outputs of P and P'; if $K = (K')^2$ then it is in the real world, otherwise it is in the ideal world (unless the independently random K' in the ideal world satisfies $K = (K')^2$, which happens with probability at most $2/p$).

The attack above can be easily prevented by hashing the Diffie–Hellman output, since $H(K')$ and $H((K')^2)$ are not correlated anymore (if H is modeled as an RO). We will see that EKE with hashed Diffie–Hellman is indeed UC-secure; however, some further subtleties will emerge in the security proof (see Section 2.3).

What does this attack entail? For simple cryptographic primitives, it is usually clear what breaking the security notion means in real life (e.g., forgery in digital signatures). Here the connection is fuzzier: does the adversary above break the security of EKE with plain Diffie–Hellman in any "actual" sense, or does it merely break some UC notion?

A philosophical answer is that one cannot talk about security clearly without a definition, and since the community has accepted the UC notion as the "right" one (see Section 2.1), UC-security literally *is* the security for PAKE, and breaking the UC-security notion *is* breaking the "actual" security — after all, what do you even mean by "actual" security if it doesn't mean UC?

This argument might be satisfactory to some but ridiculous to others. We now give a more concrete answer. This attack breaks a form of forward secrecy: P''s instance is unattacked when it is active; after P''s instance completes, when the adversary uses the password to mess with P's instance, can it still learn something about P''s session key? Forward secrecy should disallow this, yet this is exactly what the adversary achieves in the attack above. In some sense, this attack confirms that the UC notion is the "right" way to define various security requirements for PAKE, including forward secrecy.

We would also like to stress some theoretical aspects of this attack. It is an excellent example that some unexpected subtleties might arise in a security notion, and without actually attempting

a security proof, one can never be certain whether a protocol is secure or not; *relying on your intuition instead of performing an actual security analysis, or believing some "folklore" knowledge, is a recipe for disaster*. This is especially the case for security notions that are complicated, such as those in the UC framework.

2.3 Cryptographic Assumptions

We now turn to EKE with hashed Diffie–Hellman, which is the protocol that we will analyze. Below we simply call this protocol "EKE".

The Hash Diffie–Hellman (HDH) assumption. We abstract the security of the (unauthenticated) hashed Diffie–Hellman KE as the following assumption:

Definition 1. *We say the* ***Hash Diffie–Hellman (HDH) assumption*** *holds in group* $(\mathbb{G}, g, p)$ *if the following two distributions are indistinguishable:*

$x \leftarrow \mathbb{Z}_p$; $X := g^x$ $y \leftarrow \mathbb{Z}_p$; $Y := g^y$ $K := H(g^{xy})$ output (X, Y, K) to $\mathcal{A}$	$x \leftarrow \mathbb{Z}_p$; $X := g^x$ $y \leftarrow \mathbb{Z}_p$; $Y := g^y$ $K \leftarrow \{0,1\}^n$ output (X, Y, K) to $\mathcal{A}$

In the ROM where $H : \mathbb{G} \to \{0,1\}^n$ is modeled as an RO, the HDH assumption can be reduced to Computational Diffie–Hellman (CDH) via a loose reduction, which needs to make a guess among all of the HDH adversary's H queries as (the reduction's guess of) g^{xy}.

Using HDH in the UC-security of EKE. Consider the following attack on EKE, which is a generalization of the attack in Figure 3 where the adversary chooses any $Y^* \neq Y$ rather than $Y^* = Y^2$:

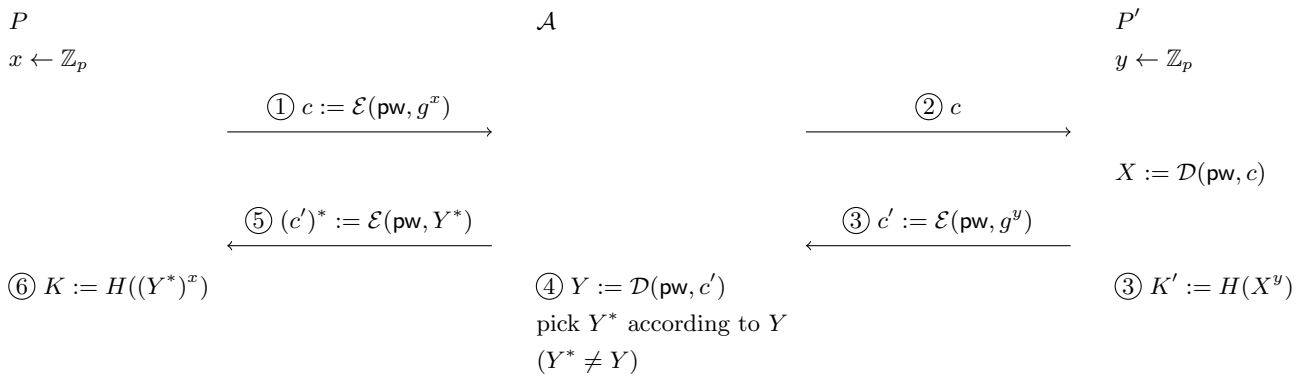


Figure 4: Attack on EKE that requires 1-ODH

By the same argument as in Section 2.2, in the ideal world K' is independent of everything else. In other words, in the ideal world K' is indistinguishable from random *even given* K. This must

hold in the real world as well, where $K' = H(g^{xy})$ and $K = H((Y^*)^x)$. (In the unhashed version this indistinguishability breaks down if $Y^* = Y^2$, which is the attack in Figure 3.) This suggests that the UC-security of EKE requires an extension of HDH which says that HDH holds *even if the adversary is additionally given* $H((Y^*)^x)$ *for any* $Y^* \neq Y$ *of the adversary's choice.* This is formalized as the following assumption.

The 1-query Oracle Diffie–Hellman (1-ODH) assumption.

Definition 2. *We say the* ***1-query Oracle Diffie–Hellman (1-ODH) assumption*** *holds in group* $(\mathbb{G}, g, p)$ *if the following two distributions are indistinguishable:*[5]

$x \leftarrow \mathbb{Z}_p$; $X := g^x$	$x \leftarrow \mathbb{Z}_p$; $X := g^x$
$y \leftarrow \mathbb{Z}_p$; $Y := g^y$	$y \leftarrow \mathbb{Z}_p$; $Y := g^y$
$K' := H(g^{xy})$	$K' \leftarrow \{0,1\}^n$
output (X, Y, K') to $\mathcal{A}$	output (X, Y, K') to $\mathcal{A}$
$Y^* \leftarrow \mathcal{A}$	$Y^* \leftarrow \mathcal{A}$
abort if $Y^* = Y$	abort if $Y^* = Y$
$K := H((Y^*)^x)$	$K := H((Y^*)^x)$
output K to $\mathcal{A}$	output K to $\mathcal{A}$

At first glance, it might appear that K is independent of K', so additionally learning K should not increase the adversary's advantage. However, in fact the reduction from 1-ODH to CDH is even looser than the HDH reduction. To see this, consider an adversary $\mathcal{A}$ that samples $r \leftarrow \mathbb{Z}_p$ and sets $Y^* := g^r$; when $\mathcal{A}$ receives K, it checks if $K = H(X^r)$ (which should hold since $X^r = g^{rx} = (Y^*)^x$) and aborts if not. The reduction must make sure that $K = H(g^{rx})$ holds, but among all of $\mathcal{A}$'s H queries, it cannot tell which one is g^{rx} — which means that the reduction has to make two guesses over $\mathcal{A}$'s H queries, one for g^{rx} and one for g^{xy}. Overall the reduction loses a factor of $\Theta(q^2)$ for q queries.[6]

A non-attack. In the above attack that requires 1-ODH, the adversary passes the first message but then modifies the second. Interestingly, the case that the adversary modifies the first message and passes the second can be simulated perfectly, without requiring 1-ODH (or even HDH). This suggests that the first and second messages should be treated differently in the security proof.

In the scenario in Figure 5, the simulator $\mathcal{S}$ can extract pw as soon as $\mathcal{A}$ sends c^*, by looking at $\mathcal{A}$'s $\mathcal{E}$ queries. Then $\mathcal{S}$ can send $(\mathsf{TestPwd}, sid, P', \mathsf{pw})$ to $\mathcal{F}_{\text{PAKE}}$ and make the P' instance $\mathsf{compromised}$, which allows $\mathcal{S}$ to set P''s key K' (which $\mathcal{S}$ can simulate honestly by sampling a random y and computing $K' := H((X^*)^y)$). Later when $\mathcal{A}$ passes c', $\mathcal{S}$ can send $(\mathsf{TestPwd}, sid, P, \mathsf{pw})$ to $\mathcal{F}_{\text{PAKE}}$, make the P instance $\mathsf{compromised}$, and set P's key K (which $\mathcal{S}$ can again simulate honestly as $K := H(X^y)$). In this way, the keys of P and P' are simulated exactly the same as in the real world.

[5] The ODH assumption [ABR01] says that $H(g^{xy})$ is indistinguishable from random given g^x, g^y and access to an additional oracle $H_x(\cdot)$ that on input $Y^* \neq g^y$ outputs $H((Y^*)^x)$. 1-ODH is ODH except that $H_x(\cdot)$ can be queried only once, hence the name. The 1-ODH assumption was introduced in [JKSS12] while studying the security of authenticated key exchange; it should naturally follow that this assumption would also emerge in the context of PAKE, yet this was unnoticed until [JRX25].

[6] This assumes that DDH is hard in the group; otherwise both HDH and 1-ODH can be tightly reduced to CDH.

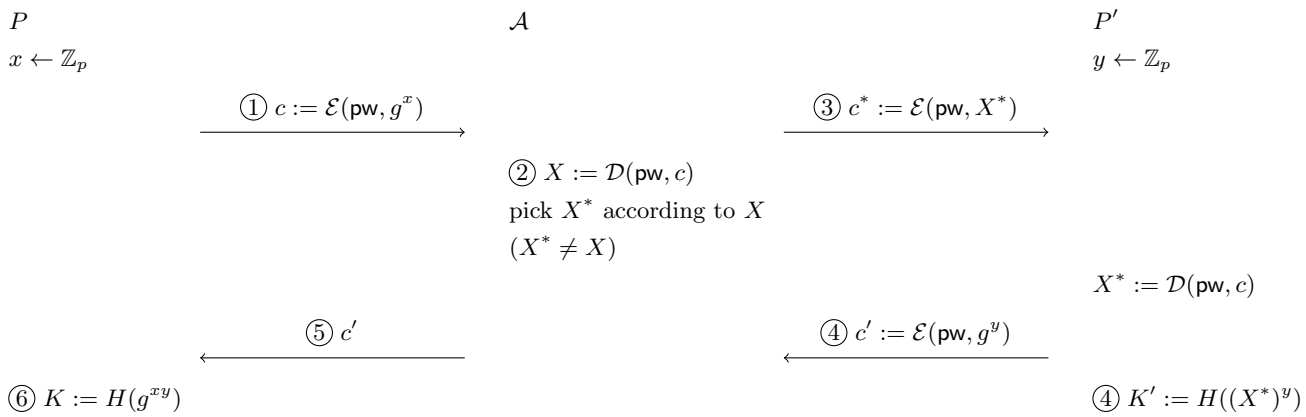


Figure 5: Non-attack on EKE that can be simulated perfectly

The crucial difference between Figures 4 and 5 lies in when exactly $\mathcal{S}$ learns pw. In Figure 4 $\mathcal{S}$ learns pw *after* P' outputs K', so the P' instance is fresh and K' is independent of $\mathcal{S}$'s view. By contrast, in Figure 5 $\mathcal{S}$ learns pw *before* P' outputs K', so P''s instance can be $\mathsf{compromised}$ by a $\mathsf{TestPwd}$ command, which allows $\mathcal{S}$ to set K'.

3 UC-Security of EKE

> [I]t is pretty widely acknowledge [sic] that universal composability is *really hard to use* in papers. [...] A colleague of mine has said that proving non-trivial things using UC is like writing a web server in assembly language.
>
> A Stack Exchange user [pg117]

3.1 UC Description of the EKE Protocol

The EKE protocol is shown in Figure 6. It is written as a non-simultaneous protocol, where P' must wait for the message from P before sending its own message; while in fact the P-to-P' and P'-to-P messages can be sent simultaneously. This minor change will make our security proof cleaner.[7]

UC conventions. We assume without loss of generality that the two parties' names are included in *sid*. For brevity, we omit *sid* in the protocol messages and output keys; for example, P' should

[7] The main benefit is to fix P' as the party who outputs first. In the 1-simultaneous version, which party outputs first depends on the order of four events, which is decided by the environment: when P starts its instance; when P' starts its instance; when the P-to-P' message is delivered to P'; and when the P'-to-P message is delivered to P. This will make the proof messier but not more difficult in any essential sense.

in fact output (sid, K') and send (sid, c') to P, but both sid are omitted. Furthermore, all RO and IC queries should include sid as part of the input, which is also omitted; for example, $\mathcal{E}(\mathsf{pw}, g^x)$ should in fact be $\mathcal{E}(sid||\mathsf{pw}, g^x)$.

The protocol uses a group $(\mathbb{G}, g, p)$, an ideal cipher $(\mathcal{E}, \mathcal{D})$ with $\mathcal{E} : \{0,1\}^n \times \mathbb{G} \to \{0,1\}^n$ and $\mathcal{D} : \{0,1\}^n \times \{0,1\}^n \to \mathbb{G}$, and a random oracle $H : \mathbb{G} \to \{0,1\}^n$.

1. On input $(\mathsf{NewSession}, sid, P, P', \mathsf{pw})$, if this is the first NewSession message for sid, party P samples $x \leftarrow \mathbb{Z}_p$, computes $c := \mathcal{E}(\mathsf{pw}, g^x)$, and sends c to P'.
2. On input $(\mathsf{NewSession}, sid, P', P, \mathsf{pw}')$ and c from P, if this is the first NewSession message for sid, party P' samples $y \leftarrow \mathbb{Z}_p$, computes $K' := H(\mathcal{D}(\mathsf{pw}', c)^y)$ and $c' := \mathcal{E}(\mathsf{pw}', g^y)$, outputs K', and sends c' to P.
3. On c' from P', party P computes $K := H(\mathcal{D}(\mathsf{pw}, c')^x)$ and outputs K.

Figure 6: Protocol EKE

Correctness can be easily verified: assuming $\mathsf{pw} = \mathsf{pw}'$ and there is no adversary, we have that

$$K = H(\mathcal{D}(\mathsf{pw}, c')^x) = H((g^y)^x) = H(g^{xy}),$$
$$K' = H(\mathcal{D}(\mathsf{pw}', c)^y) = H((g^x)^y) = H(g^{xy}).$$

3.2 Security Proof

Theorem 1. *Suppose that the CDH assumption holds in group $(\mathbb{G}, g, p)$. Then the protocol in Figure 6 UC-realizes $\mathcal{F}_{PAKE}$ (Figure 2).*

Proof. We abbreviate $\mathcal{F}_{\text{PAKE}}$ as $\mathcal{F}$. The high-level idea of the simulator $\mathcal{S}$ is as follows: The first message c is simulated as a uniformly random string. On the adversary $\mathcal{A}$'s P-to-P' message c^*,

- If $\mathcal{A}$ passes c without modification, $\mathcal{S}$ lets P' output its session key via a NewKey command to $\mathcal{F}$ (without TestPwd), and simulates the P'-to-P message c' with a uniformly random string.
- Otherwise $\mathcal{S}$ attempts to extract a password guess pw^* (implicitly) contained in c^* by observing $\mathcal{A}$'s IC encryption queries, and sends a TestPwd command on pw^* to $\mathcal{F}$. (If there is no such password guess, i.e., $\mathcal{A}$ is essentially sending random junk, $\mathcal{S}$ sends TestPwd on $\perp$.) If the password guess is correct, $\mathcal{S}$ now obtains enough information and power to simulate both P''s session key K' and the P'-to-P message c' honestly; otherwise $\mathcal{F}$ will let P' output a uniformly random session key, and $\mathcal{S}$ simulates c' with a uniformly random string.

Finally, on $\mathcal{A}$'s P'-to-P message $(c')^*$,

- If $\mathcal{A}$ is passive (i.e., it passes *both* c and c' without modification), $\mathcal{S}$ lets P output its session key via a NewKey command to $\mathcal{F}$ (without a TestPwd).
- As we have mentioned, the case in Figure 5 should be treated separately: $\mathcal{S}$ has extracted the correct password guess pw' (for P') from the first message c^*, and should send a TestPwd command on pw' in *both* directions. $\mathcal{S}$ does not know whether the two parties' passwords match when it sends TestPwd, but if so, it can simulate both parties' session keys honestly.

- Otherwise $\mathcal{S}$ attempts to extract a password guess $(\mathsf{pw}')^*$ contained in $(c')^*$, and sends a $\mathsf{TestPwd}$ command on $(\mathsf{pw}')^*$ (or $\perp$ if there is no password guess) to $\mathcal{F}$. If the password guess is correct, $\mathcal{S}$ can simulate P's session key K honestly; otherwise $\mathcal{F}$ will let P output a uniformly random session key.

1. On $(\mathsf{NewSession}, sid, P, P')$ from $\mathcal{F}$, sample $c \leftarrow \{0,1\}^n$ and send c from P to P'.
2. On $(\mathsf{NewSession}, sid, P', P)$ from $\mathcal{F}$ and c^* from $\mathcal{A}$ to P', simulate P''s output and the P'-to-P message as follows:
   (a) If $c^* = c$, send $(\mathsf{NewKey}, sid, P', 0^n)$ to $\mathcal{F}$; sample $c' \leftarrow \{0,1\}^n$ and send c' from P' to P.
   (b) If $c^* \neq c$, search for an IC encryption query $\mathcal{E}(\mathsf{pw}^*, X^*)$ whose result is c^*. // extract password guess for P'
      (i) If there is more than one such pw^*, output $\mathsf{Collision}$ and abort.
      (ii) If there is exactly one such pw^*, send $(\mathsf{TestPwd}, sid, P', \mathsf{pw}^*)$ to $\mathcal{F}$.
      If $\mathcal{F}$ replies with "correct guess", sample $y \leftarrow \mathbb{Z}_p$; send $(\mathsf{NewKey}, sid, P', H((X^*)^y))$ to $\mathcal{F}$; send $\mathcal{E}(\mathsf{pw}^*, g^y)$ from P' to P. // pw^* is equal to P''s password pw'
      If $\mathcal{F}$ replies with "wrong guess", send $(\mathsf{NewKey}, sid, P', 0^n)$ to $\mathcal{F}$; sample $c' \leftarrow \{0,1\}^n$ and send c' from P' to P.
      (iii) If there is no such pw^*, send $(\mathsf{TestPwd}, sid, P', \perp)$ and then $(\mathsf{NewKey}, sid, P', 0^n)$ to $\mathcal{F}$; sample $c' \leftarrow \{0,1\}^n$ and send c' from P' to P.
3. On $(c')^*$ from $\mathcal{A}$ to P, simulate P's output as follows:
   (a) If $c^* = c \wedge (c')^* = c'$, send $(\mathsf{NewKey}, sid, P, 0^n)$ to $\mathcal{F}$. // $\mathcal{A}$ is passive
   (b) If $\mathcal{S}$ previously entered step 2(b)(ii) and received "correct guess" (note that this implies that $c^* \neq c$), then at that time a pw^* was defined (and c' was set to $\mathcal{E}(\mathsf{pw}^*, g^y)$). If furthermore $(c')^* = c'$, send $(\mathsf{TestPwd}, sid, P, \mathsf{pw}^*)$ to $\mathcal{F}$. // pw^* is equal to P''s password pw'
      If $\mathcal{F}$ replies with "correct guess", send $(\mathsf{NewKey}, sid, P, H(\mathcal{D}(\mathsf{pw}^*, c)^y))$ to $\mathcal{F}$. // pw^* is also equal to P's password pw; this is Figure 5
      If $\mathcal{F}$ replies with "wrong guess", send $(\mathsf{NewKey}, sid, P, 0^n)$ to $\mathcal{F}$.
   (c) Otherwise search for an IC encryption query $\mathcal{E}((\mathsf{pw}')^*, Y^*)$ whose result is $(c')^*$. // extract password guess for P
      (i) If there is more than one such $(\mathsf{pw}')^*$, output $\mathsf{Collision}'$ and abort.
      (ii) If there is exactly one such $(\mathsf{pw}')^*$, send $(\mathsf{TestPwd}, sid, P, (\mathsf{pw}')^*)$ to $\mathcal{F}$.
      If $\mathcal{F}$ replies with "correct guess", compute $g^x := \mathcal{D}((\mathsf{pw}')^*, c)$ and send $(\mathsf{NewKey}, sid, P, H((Y^*)^x))$ to $\mathcal{F}$. // $(\mathsf{pw}')^*$ is equal to P's password pw
      If $\mathcal{F}$ replies with "wrong guess", send $(\mathsf{NewKey}, sid, P, 0^n)$ to $\mathcal{F}$.
      (iii) If there is no such $(\mathsf{pw}')^*$, send $(\mathsf{TestPwd}, sid, P, \perp)$ and then $(\mathsf{NewKey}, sid, P, 0^n)$ to $\mathcal{F}$.

Simulation of the RO and IC: answer H and $(\mathcal{E}, \mathcal{D})$ queries (by $\mathcal{A}$ or by $\mathcal{S}$ itself) via lazy sampling, except that for $\mathcal{D}(\star, c)$ queries, additionally store the "discrete log trapdoor" (i.e., x^* such that the answer is g^{x^*}). This is needed in step 3(c)(ii).

Figure 7: UC simulator $\mathcal{S}$ for EKE

Concretely, the simulator $\mathcal{S}$ is shown in Figure 7. As standard in UC, we assume that the

adversary $\mathcal{A}$ is "dummy" that merely passes all messages to and from the environment $\mathcal{Z}$ (see [Can01, Claim 11]). We use the following conventions: if $\mathcal{S}$ sends a message m to $\mathcal{Z}$ that pretends to be from party P to P', we abbreviate it as "send m from P to P'" — although it is actually from $\mathcal{S}$ to $\mathcal{Z}$. Similarly, if $\mathcal{S}$ receives a message m from $\mathcal{Z}$ that instructs $\mathcal{A}$ to send m to P, we abbreviate it as "on m from $\mathcal{A}$ to P".

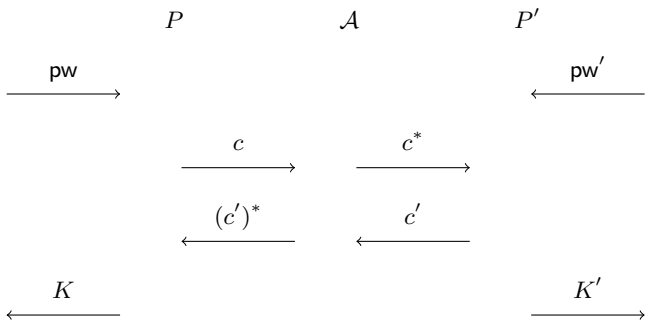
Without loss of generality, we assume that if $\mathcal{A}$ makes a $\mathcal{D}(k, c)$ query whose result is m, then it never makes a "redundant" $\mathcal{E}(k, m)$ query (it already knows that the result will be c).

Hybrids. The proof goes by a hybrid argument, which starts from the real world and ends in the ideal world. We use $\mathbf{Dist}_{\mathcal{Z}}^{i,i+1}$ to denote $\mathcal{Z}$'s distinguishing advantage between hybrids i and $i+1$. Since HDH and 1-ODH can be (loosely) reduced to CDH, in the hybrids below we may argue for indistinguishability by reducing to HDH or 1-ODH rather than CDH (although the theorem statement only assumes CDH).

At a high level, we will first deal with the "correct execution" case (i.e., $\mathcal{A}$ is passive and the two parties' passwords match) in hybrid 1. Then in hybrids 2–5 we will modify the P' side (P''s session key K' and protocol message c') to match the ideal world. After that, in hybrids 6–11 we will modify the P side to match the ideal world. On each side, we proceed case by case: we examine the simulator and identify cases where the simulation is not perfect (i.e., the ideal world is different from the real world), and in each case we make a hybrid showing that the simulator generates a different yet indistinguishable view.

Hybrid 0: This is the real world. Recall that the passwords of P and P' are pw and pw', respectively; the flow of events is:

- P sends IC ciphertext c to P';
- It is intercepted by the man-in-the-middle adversary $\mathcal{A}$, who sends c^* (which may or may not be equal to c) to P';
- P' outputs key K' and sends IC ciphertext c' to P;
- It is again intercepted by $\mathcal{A}$, who sends $(c')^*$ to P;
- P outputs key K.



Hybrid 1 (same keys if passwords match and $\mathcal{A}$ passive): In the case that $\mathsf{pw} = \mathsf{pw}' \wedge c^* = c \wedge (c')^* = c'$, set $K := K'$.

The difference between hybrids 0 and 1 is that in hybrid 0 P outputs the real key K, whereas in hybrid 1 P copies P''s key K'. By correctness of EKE (see the end of Section 3.1), P's real key K is equal to K' in hybrid 0, so this change makes no external difference. We have that

$$\mathbf{Dist}_{\mathcal{Z}}^{0,1} = 0.$$

Hybrid 2 (P' side all random if passwords don't match and $\mathcal{A}$ passes c): In the case that $\mathsf{pw} \neq \mathsf{pw}' \wedge c^* = c$, set $K', c' \leftarrow \{0,1\}^n$. [This corresponds to step 2(a) of the simulator when $\mathsf{pw} \neq \mathsf{pw}'$. We will see very soon why we need to divide step 2(a) into two sub-cases according to whether $\mathsf{pw} = \mathsf{pw}'$ or not.]

In hybrid 1, (K', c') is computed as $(H(\mathcal{D}(\mathsf{pw}', c)^y), \mathcal{E}(\mathsf{pw}', g^y))$ for $y \leftarrow \mathbb{Z}_p$. Let $X' = \mathcal{D}(\mathsf{pw}', c)$. Since $c = \mathcal{E}(\mathsf{pw}, g^x)$ and $\mathsf{pw} \neq \mathsf{pw}'$, $X' = \mathcal{D}(\mathsf{pw}', c)$ is a random group element *independent of the P side*. The maximum $\mathcal{Z}$ can do is to (instruct $\mathcal{A}$ to) decrypt c and obtain X', and decrypt c' and obtain g^y, but even then $K' = H((X')^y)$ is indistinguishable from random under the HDH assumption.

Concretely, we construct a reduction $\mathcal{R}_{\text{HDH1}}$ to HDH. The high-level analysis above suggests that $\mathcal{R}_{\text{HDH1}}$ should embed its HDH challenge as X', g^y and K'. That is, $\mathcal{R}_{\text{HDH1}}$, on input (X', Y, K'), runs the code of hybrid 1, except that if $\mathsf{pw} \neq \mathsf{pw}' \wedge c^* = c$, $\mathcal{R}_{\text{HDH1}}$ lets P' output K' and send $c' := \mathcal{E}(\mathsf{pw}', Y)$ to P; furthermore, $\mathcal{R}_{\text{HDH1}}$ defines the result of $\mathcal{D}(\mathsf{pw}', c)$ as X'. ($\mathcal{R}_{\text{HDH1}}$ can simulate the P side without knowledge of $\log X'$ or $\log Y$, exactly because $\mathsf{pw} \neq \mathsf{pw}'$ and thus the P side is independent of X' and Y. In particular, $\mathcal{R}_{\text{HDH1}}$ can sample P's exponent x itself, since x is not used on the P' side.) Finally, $\mathcal{R}_{\text{HDH1}}$ copies $\mathcal{Z}$'s output bit.

$\mathcal{R}_{\text{HDH1}}$'s challenge (X', Y, K') satisfies either $K' = H((X')^y)$ or $K' \leftarrow \{0,1\}^n$, and $\mathcal{R}_{\text{HDH1}}$ is attempting to distinguish between these two cases. If $K' = H((X')^y)$, then $K' = H(\mathcal{D}(\mathsf{pw}', c)^y)$ (because $\mathcal{R}_{\text{HDH1}}$ sets $\mathcal{D}(\mathsf{pw}', c) := X'$) and $c' = \mathcal{E}(\mathsf{pw}', Y) = \mathcal{E}(\mathsf{pw}', g^y)$, which exactly matches hybrid 1. If $K' \leftarrow \{0,1\}^n$, then the only place where $Y = g^y$ is used in the entire game is when computing $c' := \mathcal{E}(\mathsf{pw}', Y)$ (note in particular that unlike the $K' = H((X')^y)$ case, here y is not used while computing K'), so c' is a random string in $\{0,1\}^n$ independent of the rest of the game, which exactly matches hybrid 2. We have that

$$\mathbf{Dist}_{\mathcal{Z}}^{1,2} = \mathbf{Adv}_{\mathcal{R}_{\text{HDH1}}}^{\text{HDH}},$$

where $\mathbf{Adv}_{\mathcal{R}_{\text{HDH1}}}^{\text{HDH}}$ is $\mathcal{R}_{\text{HDH1}}$'s distinguishing advantage in the HDH game (same below).

Hybrid 3 (P' side all random if passwords match and $\mathcal{A}$ passes c): In the case that $\mathsf{pw} = \mathsf{pw}' \wedge c^* = c$, sample $K', c' \leftarrow \{0,1\}^n$. [This corresponds to step 2(a) of the simulator when $\mathsf{pw} = \mathsf{pw}'$.]

The argument under hybrid 2 does not work anymore, since here $X = \mathcal{D}(\mathsf{pw}, c) = \mathcal{D}(\mathsf{pw}', c)$ might be used on *both the P side and the P' side*. This means that we need the following property: given $X := g^x$ and $Y := g^y$ (for $x, y \leftarrow \mathbb{Z}_p$), $K' = H(g^{xy})$ is indistinguishable from random *even if we take what happens on the P side into account*. What happens on the P side is: $\mathcal{A}$ can send $(c')^* = \mathcal{E}(\mathsf{pw}, Y^*)$ for Y^* of its choice, and additionally learn P's output $K = H((Y^*)^x)$ — which is the 1-ODH assumption.

Concretely, we construct a reduction $\mathcal{R}_{\text{ODH}}$ to 1-ODH. $\mathcal{R}_{\text{ODH}}$, on input (X, Y, K'), runs $\mathcal{Z}$ (which chooses password pw for P and pw' for P') and does the following:

- $\mathcal{R}_{\text{ODH}}$ sends $c := \mathcal{E}(\mathsf{pw}, X)$ from P to P'.

- On c^* from P to P', if $\mathsf{pw} = \mathsf{pw}' \wedge c^* = c$ does not hold, $\mathcal{R}_{\text{ODH}}$ aborts. Otherwise $\mathcal{R}_{\text{ODH}}$ lets P' output K' and send $c' := \mathcal{E}(\mathsf{pw}, Y)$ to P.
- On $(c')^*$ from P' to P, if $(c')^* = c'$, then $\mathcal{R}_{\text{ODH}}$ sends an arbitrary $Y' \neq Y$ to the 1-ODH challenger (just to finish the 1-ODH game) and lets P output K'. Otherwise $\mathcal{R}_{\text{ODH}}$ sets $Y^* := \mathcal{D}(\mathsf{pw}, (c')^*)$; by the definition of IC, $Y^* \neq Y$ (otherwise $\mathcal{E}(\mathsf{pw}, Y) = \mathcal{E}(\mathsf{pw}, Y^*)$ is both c' and $(c')^*$), so $\mathcal{R}_{\text{ODH}}$ sends Y^* to the 1-ODH challenger and receives K, and lets P output K.

Finally, $\mathcal{R}_{\text{ODH}}$ copies $\mathcal{Z}$'s output bit.

Suppose $\mathsf{pw} = \mathsf{pw}' \wedge c^* = c$. Then if $K' = H(g^{xy})$, then $\mathcal{R}_{\text{ODH}}$ simulates hybrid 2; if $K' \leftarrow \{0,1\}^n$, then $\mathcal{R}_{\text{ODH}}$ simulates hybrid 3. This can be seen via the following:

- In both hybrids 2 and 3, P computes its protocol message $c := \mathcal{E}(\mathsf{pw}, g^x)$, whereas here $\mathcal{R}_{\text{ODH}}$ computes $c := \mathcal{E}(\mathsf{pw}, X)$ where X is from the 1-ODH challenger. This does not make any external difference.
- On c, in hybrid 2 P' decrypts $X := \mathcal{D}(\mathsf{pw}, c)$ and outputs $K' := H(X^y) = H(g^{xy})$, whereas in hybrid 3 P' outputs $K' \leftarrow \{0,1\}^n$. Again this matches what $\mathcal{R}_{\text{ODH}}$ does: it lets P' output K' from the 1-ODH challenger, which is either $H(g^{xy})$ or a random group element.
- For the P'-to-P message c', in hybrid 2 it is defined as $c' := \mathcal{E}(\mathsf{pw}, g^y)$, whereas in hybrid 3 it is defined as $c' \leftarrow \{0,1\}^n$. $\mathcal{R}_{\text{ODH}}$ computes $c' := \mathcal{E}(\mathsf{pw}, Y)$ for Y from the 1-ODH challenger, which matches hybrid 2. Furthermore, if $K' \leftarrow \{0,1\}^n$, then the only place where $Y = g^y$ is used in the entire game is in computing $c' := \mathcal{E}(\mathsf{pw}, Y)$ (note in particular that y is not used while computing K'), so c' is a random string in $\{0,1\}^n$ independent of the rest of the game, which matches hybrid 3. In sum, if $K' = H(g^{xy})$ then $\mathcal{R}_{\text{ODH}}$ computes c' as in hybrid 2, and if $K' \leftarrow \{0,1\}^n$ then $\mathcal{R}_{\text{ODH}}$ computes c' as in hybrid 3.
- If $(c')^* = c'$, then hybrid 1 lets P output $K := K'$, which is exactly what $\mathcal{R}_{\text{ODH}}$ does here. (This shows that hybrid 3 must be done after hybrid 1; otherwise $\mathcal{R}_{\text{ODH}}$ does not know how to simulate $K = H(g^{xy})$ correctly, since K' might be a random group element.)
- If $(c')^* \neq c'$, then in both hybrids 2 and 3, P decrypts $Y^* := \mathcal{D}(\mathsf{pw}, (c')^*)$ and outputs $K := H((Y^*)^x)$, which is again what $\mathcal{R}_{\text{ODH}}$ does here.

We conclude that

$$\mathbf{Dist}_{\mathcal{Z}}^{2,3} = \mathbf{Adv}_{\mathcal{R}_{\text{ODH}}}^{\text{1-ODH}}.$$

Hybrids 2 and 3 combined cover the $c^* = c$ case. We now consider the case that $c^* \neq c$. Given $c^* \neq c$, we say c^* contains password guess pw^* for P' if there is an IC encryption query $\mathcal{E}(\mathsf{pw}^*, X^*)$ whose result is c^*.

Hybrid 4 (abort if c^* contains more than one password guess): If c^* contains more than one password guess, output Collision and abort. (In all subsequent hybrids, we assume that Collision does not happen.) [This corresponds to step 2(b)(i) of the simulator.]

Collision means that there are two different queries to $\mathcal{E}$ that both result in c^*. Assuming there are q queries to $\mathcal{E}$, by the birthday bound,

$$\mathbf{Dist}_{\mathcal{Z}}^{3,4} = \Pr[\mathsf{Collision}] \leq \frac{q(q-1)}{2^{n+1}}.$$

(Note that here we rely on the assumption that $\mathcal{A}$ never makes "redundant" $\mathcal{E}$ queries; otherwise $\mathcal{A}$ can easily trigger $\mathsf{Collision}$ by querying $X_1^* := \mathcal{D}(\mathsf{pw}_1, c^*), X_2^* := \mathcal{D}(\mathsf{pw}_2, c^*)$ for different $\mathsf{pw}_1, \mathsf{pw}_2$ and then querying $\mathcal{E}(\mathsf{pw}_1, X_1^*), \mathcal{E}(\mathsf{pw}_2, X_2^*)$.)

Hybrid 5 (P' side all random if c^* doesn't contain correct password guess): If c^* does not contain pw' as the password guess, sample $K', c' \leftarrow \{0,1\}^n$. [This corresponds to step 2(b)(ii), "wrong guess" sub-case, together with step 2(b)(iii), of the simulator.]

The critical difference between c^* contains and does not contain password guess pw' is as follows. If c^* contains password guess pw', then $\mathcal{A}$ has queried $c^* = \mathcal{E}(\mathsf{pw}', X^*)$ for some X^* of its choice, so $\mathcal{Z}$ might know $\log X^*$. However, if c^* does not contain password guess pw', then the maximum $\mathcal{Z}$ can do is to query $\mathcal{D}(\mathsf{pw}', c^*)$ and obtain a random group element X^* (*without knowing its discrete log*), plus query $\mathcal{D}(\mathsf{pw}', c')$ and obtain Y (again, without knowing its discrete log y); then P''s output $K' = H((X^*)^y)$ is indistinguishable from random under the HDH assumption. We can construct a reduction $\mathcal{R}_{\text{HDH2}}$ similar to hybrid 2, which is omitted. We have that

$$\mathbf{Dist}_{\mathcal{Z}}^{4,5} = \mathbf{Adv}^{\text{HDH}}_{\mathcal{R}_{\text{HDH2}}}.$$

Comparison between hybrid 5 and the ideal world. Let us pause a bit and compare hybrid 5 and the ideal world. We claim that hybrid 5 has changed everything on the P' side to be exactly the same as the ideal world. This can be seen from the following table.

case	K', c' definitions	in hybrid 5	in ideal world
$c^* = c$	$K', c' \leftarrow \{0,1\}^n$	changed in hybrids 2,3	step 2(a)
$c^* \neq c$, contains correct password guess	$K' := H((X^*)^y)$, $c' := \mathcal{E}(\mathsf{pw}', g^y)$	unchanged from real world	step 2(b)(ii)
$c^* \neq c$, contains wrong password guess	$K', c' \leftarrow \{0,1\}^n$	changed in hybrid 5	step 2(b)(ii)
$c^* \neq c$, contains no password guess	$K', c' \leftarrow \{0,1\}^n$	changed in hybrid 5	step 2(b)(iii)

The above cases cover all possibilities on the P' side. In the subsequent hybrids, we will change the P side to be identical to the ideal world.

Hybrid 6 (random K if passwords don't match and $\mathcal{A}$ passive): In the case that $\mathsf{pw} \neq \mathsf{pw}' \wedge c^* = c \wedge (c')^* = c'$, sample $K \leftarrow \{0,1\}^n$. [This corresponds to step 3(a) of the simulator when $\mathsf{pw} \neq \mathsf{pw}'$. Note that the $\mathsf{pw} = \mathsf{pw}'$ case was already handled in hybrid 1.]

Let's recall what hybrid 5 does in the case that $\mathsf{pw} \neq \mathsf{pw}' \wedge c^* = c \wedge (c')^* = c'$. P sends $c := \mathcal{E}(\mathsf{pw}, g^x)$ for $x \leftarrow \mathbb{Z}_p$; then P' outputs $K' \leftarrow \{0,1\}^n$ and sends $c' \leftarrow \{0,1\}^n$ (note that the P' side was already changed to all random in hybrid 2); finally P decrypts $Y^* := \mathcal{D}(\mathsf{pw}, c')$ and outputs $K := H((Y^*)^x)$ (here we need $\mathsf{pw} \neq \mathsf{pw}'$, since otherwise K is defined to be equal to K' per hybrid 1). The maximum $\mathcal{Z}$ can do is to decrypt c and obtain g^x, and decrypt c' and obtain Y^*, but even then $K = H((Y^*)^x)$ is indistinguishable from random under the HDH assumption. We can construct a reduction $\mathcal{R}_{\text{HDH3}}$ similar to hybrid 2, which is omitted. We have that

$$\mathbf{Dist}_{\mathcal{Z}}^{5,6} = \mathbf{Adv}^{\text{HDH}}_{\mathcal{R}_{\text{HDH3}}}.$$

Note that we have handled all cases where $\mathcal{A}$ is passive (i.e., $c^* = c \wedge (c')^* = c'$): if $\mathsf{pw} = \mathsf{pw}'$ then it was covered in hybrid 1, and if $\mathsf{pw} \neq \mathsf{pw}'$ then it was covered in hybrid 6.

Hybrid 7 (compute K without using x in Figure 5): In the case that $\mathsf{pw} = \mathsf{pw}' \wedge c^* \neq c \wedge (c')^* = c'$, and c^* contains the correct password guess pw' (for P'), compute $K := H(\mathcal{D}(\mathsf{pw}, c)^y)$. [This corresponds to step 3(b), "correct guess" sub-case of the simulator.]

In hybrid 6 K is computed as $H(\mathcal{D}(\mathsf{pw}, (c')^*)^x) = H(\mathcal{D}(\mathsf{pw}, c')^x) = H((g^y)^x)$; here K is computed as $H(\mathcal{D}(\mathsf{pw}, c)^y) = H((g^x)^y)$. Both are equal to $H(g^{xy})$, so there is no difference. We have that

$$\mathbf{Dist}_{\mathcal{Z}}^{6,7} = 0.$$

Hybrid 8 (random K if $\mathcal{A}$ proceeds as in Figure 5 except that passwords don't match): In the case that $\mathsf{pw} \neq \mathsf{pw}' \wedge c^* \neq c \wedge (c')^* = c'$, and c^* contains the correct password guess pw' (for P'), sample $K \leftarrow \{0,1\}^n$. [This corresponds to step 3(b), "wrong guess" sub-case of the simulator.]

In this case hybrid 7 proceeds as follows: P sends $c := \mathcal{E}(\mathsf{pw}, g^x)$ for $x \leftarrow \mathbb{Z}_p$; then P' outputs $K' := H((X^*)^y)$ and sends $c' := \mathcal{E}(\mathsf{pw}', g^y)$ for $y \leftarrow \mathbb{Z}_p$; finally P decrypts $Y^* := \mathcal{D}(\mathsf{pw}, c')$ and outputs $K := H((Y^*)^x)$. The argument is similar to hybrid 6: the maximum $\mathcal{Z}$ can do is to decrypt c and obtain g^x, and decrypt c' and obtain Y^*, but even then $K = H((Y^*)^x)$ is indistinguishable from random under the HDH assumption. Crucially, since $\mathsf{pw} \neq \mathsf{pw}'$, $g^y = \mathcal{D}(\mathsf{pw}', c')$ is independent of $Y^* = \mathcal{D}(\mathsf{pw}, c')$ and thus the entire P' side can be simulated honestly by the HDH reduction $\mathcal{R}_{\text{HDH4}}$ who samples y on its own (because y is not used on the P side). We have that

$$\mathbf{Dist}_{\mathcal{Z}}^{7,8} = \mathbf{Adv}^{\text{HDH}}_{\mathcal{R}_{\text{HDH4}}}.$$

Below we consider all remaining cases, which contain the following sub-cases:

1. $c^* = c \wedge (c')^* \neq c'$;
2. $c^* \neq c$, and c^* does not contain the correct password guess pw' (for P');
3. $c^* \neq c \wedge (c')^* \neq c'$, and c^* contains the correct password guess pw' (for P').

(These three sub-cases combined cover all possibilities except the following: $\mathcal{A}$ is passive, which was covered in hybrids 1 and 6; $c^* \neq c \wedge (c')^* = c'$, and c^* contains the correct password guess pw', which is covered in hybrids 7 and 8.)

In cases 1–3, we say $(c')^*$ contains password guess $(\mathsf{pw}')^*$ for P if there is an IC encryption query $\mathcal{E}((\mathsf{pw}')^*, Y^*)$ whose result is $(c')^*$.

Hybrid 9 (abort if $(c')^*$ contains more than one password guess): If $(c')^*$ contains more than one password guess, output $\mathsf{Collision}'$ and abort. (In all subsequent hybrids, we assume that $\mathsf{Collision}'$ does not happen.) [This corresponds to step 3(c)(i) of the simulator.]

By an argument similar to hybrid 4,

$$\mathbf{Dist}_{\mathcal{Z}}^{7,8} = \Pr[\mathsf{Collision}'] \leq \frac{q(q-1)}{2^{n+1}}.$$

Hybrid 10 (random K if $(c')^*$ doesn't contain correct password guess): If $(c')^*$ does not contain pw as the password guess, sample $K \leftarrow \{0,1\}^n$. [This corresponds to step 3(c)(ii), "wrong guess" sub-case, together with step 3(c)(iii), of the simulator.]

We construct a reduction $\mathcal{R}_{\text{HDH5}}$ to HDH; the key point is to make sure that the P side is independent of the P' side (otherwise we need 1-ODH, like in hybrid 3). We consider case 3 first. In this case hybrid 9 proceeds as follows: P sends $c := \mathcal{E}(\mathsf{pw}, g^x)$ for $x \leftarrow \mathbb{Z}_p$, which is modified to

c^* by $\mathcal{A}$; then P' decrypts $X^* := \mathcal{D}(\mathsf{pw}', c^*)$, outputs $K' := H((X^*)^y)$ and sends $c' := \mathcal{E}(\mathsf{pw}', g^y)$ for $y \leftarrow \mathbb{Z}_p$, which is again modified to $(c')^*$ by $\mathcal{A}$; finally P decrypts $Y^* := \mathcal{D}(\mathsf{pw}, (c')^*)$ and outputs $K := H((Y^*)^x)$. The argument that K is indistinguishable from random is similar to hybrid 8:

- In hybrid 8, $(c')^* = c'$ but $\mathsf{pw} \neq \mathsf{pw}'$, so $g^y = \mathcal{D}(\mathsf{pw}', c')$ on the P' side is independent of $Y^* = \mathcal{D}(\mathsf{pw}, (c')^*)$ on the P side;
- Here, since $(c')^*$ does not contain pw as the password guess, $\mathcal{A}$ does not make an $\mathcal{E}(\mathsf{pw}, Y^*)$ query; since $(c')^* \neq c'$, the honest P' also does not make an $\mathcal{E}(\mathsf{pw}, Y^*)$ query (its IC encryption query $\mathcal{E}(\mathsf{pw}', g^y)$ results in c'). Therefore, $Y^* = \mathcal{D}(\mathsf{pw}, (c')^*)$ on the P side is a freshly sampled random group element and thus independent of the P' side.

Once we make sure that the P side and the P' side are independent, the rest of the argument is identical to hybrid 8. For cases 1 and 2, the argument is easier: in case 1 $c^* \neq c$, so the P' side is all random (see hybrids 2 and 3), which is of course independent of the P side; in case 2 c^* does not contain the correct password guess pw', so the P' side is again all random (see hybrid 5) and independent of the P side. We conclude that in all cases 1–3 the P side and the P' side are independent, so the HDH reduction $\mathcal{R}_{\text{HDH5}}$ goes through. We have that

$$\mathbf{Dist}_{\mathcal{Z}}^{9,10} = \mathbf{Adv}^{\text{HDH}}_{\mathcal{R}_{\text{HDH5}}}.$$

Comparison between hybrid 10 and the ideal world. We claim that P's session key K in hybrid 10 is exactly the same as the ideal world. This can be seen from the following table.

case	K definition	in hybrid 10	in ideal world
$\mathsf{pw} = \mathsf{pw}' \wedge c^* = c \wedge (c')^* = c'$	$K := K'$	changed in hybrid 1	step 3(a)
$\mathsf{pw} \neq \mathsf{pw}' \wedge c^* = c \wedge (c')^* = c'$	$K \leftarrow \{0,1\}^n$	changed in hybrid 6	step 3(a)
$\mathsf{pw} = \mathsf{pw}' \wedge c^* \neq c \wedge (c')^* = c'$; c^* contains correct password guess (Figure 5)	$K := H(\mathcal{D}(\mathsf{pw}, c)^y)$	changed in hybrid 7	step 3(b)
$\mathsf{pw} \neq \mathsf{pw}' \wedge c^* \neq c \wedge (c')^* = c'$; c^* contains correct password guess	$K \leftarrow \{0,1\}^n$	changed in hybrid 8	step 3(b)
Else, $(c')^*$ contains correct password guess	$K := H((Y^*)^x)$	unchanged from real world	step 3(c)(ii)
Else, $(c')^*$ contains wrong password guess	$K \leftarrow \{0,1\}^n$	changed in hybrid 10	step 3(c)(ii)
Else, $(c')^*$ contains no password guess	$K \leftarrow \{0,1\}^n$	changed in hybrid 10	step 3(c)(iii)

The above cases cover all possibilities on the P side.

We have argued that the P' side has been changed to be identical to the ideal world, and so has P's session key. The only remaining parts where hybrid 10 and the ideal world differ are the P-to-P' message c and how IC decryption queries are answered.

Hybrid 11 (random c and store discrete log trapdoor upon IC decryption query): Sample $c \leftarrow \{0,1\}^n$; if $\mathcal{D}(\mathsf{pw}, c)$ is queried, define $\mathcal{D}(\mathsf{pw}, c) := g^x$ for random $x \leftarrow \mathbb{Z}_p$ and store x; when P outputs its session key, in the "$(c')^*$ contains correct password guess" case, use x to compute K:

$K := H(\mathcal{D}(\mathsf{pw}, (c')^*)^x)$. (Note that this is the only case in hybrid 10 where x is actually used later in the game.) On a $\mathcal{D}(k, c)$ query for $k \neq \mathsf{pw}$, also store the corresponding "discrete log trapdoor" x^* such that $\mathcal{D}(k, c) = g^{x^*}$.

In hybrid 10, c is defined as $\mathcal{E}(\mathsf{pw}, g^x)$; here we sample c at random first, and then define $\mathcal{D}(\mathsf{pw}, c)$ as g^x if necessary. Of course there is no difference, so we have that

$$\mathbf{Dist}_{\mathcal{Z}}^{10,11} = 0.$$

The only difference between hybrid 11 and the ideal world is that the game challenger in hybrid 11 is split into the functionality $\mathcal{F}$ and the simulator $\mathcal{S}$ in the ideal world, which makes no external difference. We conclude that hybrid 11 is identical to the ideal world in $\mathcal{Z}$'s view. The sequence of hybrids above shows that hybrid 0 (the real world) and hybrid 11 (the ideal world) are indistinguishable, which completes the proof. □

Discussion. The complication in this proof mainly lies in the fact that there are a large number of cases to consider, which depend on the following factors:

- Whether the two parties' passwords match;
- Whether the adversary modifies the first message, and if so, whether the modified message contains a correct password guess;
- Whether the adversary modifies the second message, and if so, whether the modified message contains a correct password guess.

In the vast majority of cases, either the simulation is perfect, or the P side and the P' side are independent of each other (and a reduction to HDH suffices). The only two exceptions are

1. The two parties' passwords match, and the adversary is passive: then the two parties' keys also match by the correctness of EKE (handled in hybrid 1);
2. The two parties' passwords match, the adversary passes the first message without modification, and then modifies the second message which contains a correct password guess (i.e., Figure 4): this is the most complicated case, and we need to consider both sides as a whole — hence the reduction to 1-ODH (handled in hybrid 3).

However, without carefully thinking through all possible cases, it is very hard to figure out which of them are the "special" ones, and what exact assumption we need in those cases.

4 Flaws in Existing Works

> Errors often emerge whenever one handwaves the details rather than actually work them out rigorously. [...] [P]hrases like "it is easy to see that X" are a good place to double-check when going over an argument.
>
> Oded Goldreich [Gol12]

As mentioned in Section 1.1, the security statements for EKE in several existing works are incorrect.

It would be an interesting exercise to see where exactly their security *proofs* break down. We will show flaws in four works; along the way, we will draw some general principles about security proofs in UC and/or for PAKE.

Flaw in [MRR20]. The most obvious error probably lies in [MRR20]. [MRR20, Theorem 10] claims the UC-security of a generalized version of EKE, where we take *any* 1-round key exchange protocol KE (with some appropriate properties, which are satisfied by both plain Diffie–Hellman and hashed Diffie–Hellman) and use $\mathcal{E}(\mathsf{pw}, \mathsf{KE}$ protocol message) as EKE protocol messages.[8] This would imply that the EKE with plain Diffie–Hellman is UC-secure, which, as we have seen in Section 2.2, is not the case. Where does the proof go wrong?

The reason is simple and rather non-technical. The proof sketch in [MRR20, Section 4.2] only covers two cases, "corrupt P_1" and "corrupt P_2"; in other words, it only covers the cases where the adversary *completely disregards one protocol party and only interacts with the other* (as a "corrupt protocol party"). The detailed proof in [MRR20, Appendix C.2] also only considers these two possibilities. In "normal" two-party computation, we usually assume an authenticated channel between the two protocol parties, so it suffices to assume that one party is honest and the other is corrupt. However, PAKE is in the *password-only setting* where such channel does not exist; indeed, the most complicated case is that the man-in-the-middle adversary passes the first message and modifies the second, which is completely missing in [MRR20].

> *The first lesson: For the security of PAKE, we must consider a man-in-the-middle adversary that may see and modify (or pass without modification) protocol messages in both directions.*

Flaw in [PZ23]. The second work we analyze is [PZ23], whose main contribution is the *game-based* security proof for EKE (generalized in a flavor similar to [MRR20]); however, [PZ23, Appendix C] does claim UC-security, although there is only a sketch of the simulator (and the hybrids are missing). The simulator says

> *Upon server* S *receiving* e_1 *from* $\mathcal{A}$ *on behalf of* U *with session id* ssid. $\mathcal{S}$ *does the following checks:*
>
> - *If there exists* pw *such that* $(\mathsf{pw}, \mathsf{pk}, e_1, \mathsf{enc}) \in \mathcal{L}_1$, $\mathcal{S}$ *sends* $(\mathsf{TestPW}, \mathsf{ssid}, \mathsf{S}, \mathsf{pw})$ *to* $\mathcal{F}_{\mathsf{pake}}$. *If* $\mathcal{F}_{\mathsf{pake}}$ *replies "correct", then this means that* $\mathcal{S}$ *performs a successful online attack and recover [sic]* $\mathsf{pw}_{\mathsf{U},\mathsf{S}}$. *[...] In this case,* $\mathcal{S}$ *honestly generates* e_2 *and computes the session key* SK *as we did in Lines 25 to 31 in Figure 14. Then,* $\mathcal{S}$ *sends* e_2 *to* $\mathcal{A}$ *and sends* $(\mathsf{NewKey}, \mathsf{ssid}, \mathsf{S}, \mathsf{SK})$ *to* $\mathcal{F}_{\mathsf{pake}}$ *(which can correctly set up the session key).*
> - *In other cases, then* $\mathcal{S}$ *sends* $(\mathsf{NewKey}, \mathsf{ssid}, \mathsf{S}, \mathsf{SK})$ *to* $\mathcal{F}_{\mathsf{pake}}$, *where* SK *is sampled uniformly at random.*[9]

[8] Another difference is that the encryption mechanism is a more efficient version of IC called programmable-once public function, but this is inconsequential to our discussion.

[9] S here is the protocol party called "server", which corresponds to our P'; $\mathcal{S}$ is the UC simulator. If a protocol party is named S, perhaps a better notation for the simulator would be SIM.

This corresponds to step 2(b) in our Figure 7. Here, the simulator tries to extract a password guess by looking at the IC encryption queries. If there is one, it sends a TestPwd command on this password guess, and if the answer is "correct guess", the simulator gains all information and power to simulate both P''s session key K' and the P'-to-P message c' honestly; in all other cases, K' and c' are uniformly random.

Unfortunately, this simulator does not (quite) work. A minor issue is that it does not say what the simulator should do if it extracts more than one password guess, in which case it should abort — which is our step 2(b)(i). A more serious problem is that *their simulator does not distinguish between (1) the adversary passing the P-to-P' message without modification, and (2) the adversary sending a modified message to P' that contains no password guess.* Consider the following two scenarios:

1. $\mathcal{Z}$ lets P and P' start on the same password, and instructs $\mathcal{A}$ to be passive (i.e., pass both messages without any modification);
2. $\mathcal{Z}$ lets P and P' start on the same password, and instructs $\mathcal{A}$ to send random junks in both directions.

In case 1 P and P' output the *same* random key, whereas in case 2 they output *independent* random keys; thus, the simulation strategies in these two cases must be different. The right way to do it is

1. $\mathcal{S}$ sends NewKey for both P and P', resulting in both instances being fresh and outputting the same key — which is our steps 2(a) and 3(a);
2. $\mathcal{S}$ sends TestPwd *on* $\perp$ *and then* NewKey for both P and P', resulting in both instances being interrupted and outputting independent keys — which is our steps 2(b)(iii) and 3(c)(iii).

In the [PZ23] simulator, both cases above are covered in the "In other cases" bullet, where the simulator sends NewKey without TestPwd; therefore, in case 2 $\mathcal{F}_{\text{PAKE}}$ will let both parties output the same key even though the adversary sends random junks in both directions, which clearly does not match the real world.

> *The second lesson: Before writing down a UC proof, one must fully understand the ideal functionality. For example, in UC PAKE one must understand what the states* fresh, compromised *and* interrupted *mean, as well as how the session keys depend on them.*

Another issue is — as we have mentioned — that [PZ23] does not present a hybrid argument that the ideal world and the real world are indistinguishable. This is especially problematic because the UC-security notion is not merely an extension of the game-based one; as discussed in Section 2.1, UC-security implies various security requirements such as forward secrecy which the (standard) game-based security definition does not cover. Without actually giving a UC-security proof, it is impossible to be fully convinced that the protocol is indeed UC-secure.[10]

[10] [PZ23, Section 1] stresses that their protocol has forward secrecy. However, their actual game-based security definition and security proof do not mention (or imply) forward secrecy.

The third lesson: Game-based PAKE and UC PAKE are very different security notions; it is dangerous to give a game-based security proof and then say it "extends" to UC.

Flaw in [DHP+18]. We now move on to [DHP+18]. The main topic of this work is unrelated to our point here, but as a side contribution, [DHP+18, Theorem 6] gives a UC-security proof for EKE with hashed Diffie–Hellman — the protocol we analyzed.[11] While the theorem statement is correct, it misses the reduction to 1-ODH. Game $\mathbf{G}_9$ in the proof says

> *$\mathcal{F}$ now generates a random session key upon a first* NewKey *query for an honest party P_i with fresh record (P_i, pw_i) where the other party is also honest, if (at least) one of the following events happens: [...] No output was sent to the other party yet.*

Their notations are quite different from ours; translating to our terms, this means

> *In the case that the adversary $\mathcal{A}$ passes the P-to-P' message without modification (i.e., $c^* = c$), P' now outputs a uniformly random session key $K' \leftarrow \{0,1\}^n$.*

This is the combination of our hybrids 2 and 3; recall that in hybrid 2 (the two parties' passwords do not match), a reduction to HDH suffices, whereas in hybrid 3 (the two parties' passwords match) we must reduce to 1-ODH. However, [DHP+18, Theorem 6] does a single reduction to CDH, which only has a $\Theta(q)$ loss; this means that hybrid 3 is essentially missing.

The problem is that the proof thinks the $c^* = c$ case is similar to another case, while actually they are not. The proof under $\mathbf{G}_9$ is handwavy; it says "We only sketch the proof since it is similar to the proof of Lemma 12". Lemma 12 is under another game $\mathbf{G}_5$, which (translated to our terms) says that if $\mathcal{A}$ modifies the P'-to-P message $c' = \mathcal{E}(\mathsf{pw}', g^y)$ to another $(c')^* = \mathcal{E}((\mathsf{pw}')^*, \star)$, then P outputs a random session key if $(\mathsf{pw}')^* \neq \mathsf{pw}$ — which is our hybrid 10. One does not need to do a full reduction to see the difference between hybrids 3 and 10: *in hybrid 10, since $(\mathsf{pw}')^* \neq \mathsf{pw}$, $\mathcal{A}$ does not make an IC encryption query whose result is $(c')^*$; since $(c')^* \neq c'$, the honest P' also does not make such a query. Thus, it is guaranteed that P's decryption process $\mathcal{D}(\mathsf{pw}, (c')^*)$ is independent of g^y on the P' side; whereas in hybrid 3 the P side and the P' side might be correlated.* Thus, these two cases must be handled separately.

The four lesson: Before declaring two cases are similar and thus the second case does not need a detailed argument, one must thoroughly check that this is indeed true.

Flaw in [BCP+23]. Finally we will study [BCP+23], which gives another UC-security proof for generalized EKE (similar to [MRR20]). The proof of [BCP+23, Theorem 1] also misses the reduction to 1-ODH; game $\mathbf{G}_{6.1}$ says

[11] [DHP+18] considers EKE with an additional "label", which is inconsequential.

> *On Bob's side: Upon receiving* **Epk** *from an honest Alice, instead of setting* $SK \leftarrow H(\mathsf{ssid}, P_i, P_j, \mathbf{Epk}, \mathbf{Ec}, K)$*, if* $\mathtt{SamePwd}(\mathsf{ssid}, P_i, P_j) = \mathsf{true}$*, one sets* $K' \leftarrow H^*_K(\mathsf{ssid}, \mathsf{success})$ *[...] and updates the definition* $SK \leftarrow H(\mathsf{ssid}, P_i, P_j, \mathbf{Epk}, \mathbf{Ec}, K')$.

This is even further away from our notations than [DHP+18], since it assumes a general KE protocol. In our terms (and taking the specific Diffie–Hellman KE), this means

> *On the* P' *side, upon receiving* c *from* P *(passed by the adversary* $\mathcal{A}$ *without modification, i.e.,* $c^* = c$*), instead of setting* P'*'s session key as* $H(c, c', g^{xy})$*, if the passwords of* P *and* P' *are equal, one sets* P'*'s session key as* $H(c, c', k')$ *where* $k' \leftarrow \mathbb{G}$.

The subsequent argument says "we can simply successively replace $[(g^x, g^y, g^{xy})]$ with $[(g^x, g^y, k')]$, using [the DDH assumption]".[12] While it is true that a reduction to DDH can replace g^{xy} with a random k' *while simulating* P'*'s session key*, this is not the whole picture: as we have seen in Figure 4 and then in hybrid 3, *the reduction must continue simulating the* P *side even after* P'*'s instance completes, which may depend on* x *that is unknown to the reduction.* In particular, if $\mathcal{A}$ samples $r \leftarrow \mathbb{Z}_p$ and sends $(c')^* := \mathcal{E}(\mathsf{pw}, g^r)$ to P, P will output $K = H(g^{rx})$ which allows for the environment (who knows r and can decrypt $g^x := \mathcal{D}(\mathsf{pw}, c)$) to query $H(g^{rx})$ and check for consistency with K; the reduction (who knows g^r, g^x but neither r nor x — in particular, the reduction embeds its challenge X as g^x) cannot tell which H query is on g^{rx} and thus needs to make a guess over all queries. It appears that [BCP+23] thinks the reduction is done once P' outputs its session key, causing this mistake.

> *The fifth lesson: A reduction loses some information while simulating the security game to the adversary (due to embedding the challenge into parts of the security game), and one must go through the entire reduction and make sure that the "secret information" is not needed somewhere other than our main focus.*

Why does it matter? From the discussion above, it should be clear that existing security proofs for (the UC-security of) EKE are technically incorrect; the remaining question is how serious these flaws are — which is inherently subjective.

One might dismiss the flaws as insignificant due to the following reasons:

1. The attacks merely break the UC-security w.r.t. a specific functionality, or show some tightness bounds are incorrect, rather than have any "real" consequences;
2. It is common practice to hash the Diffie–Hellman output key, so the attack on EKE with plain Diffie–Hellman (i.e., with unhashed output key) has no practical relevance.

We addressed the first argument in Section 2.2: the attack on EKE with plain Diffie–Hellman breaks its forward secrecy. In fact, we believe the best way to understand the "meaning" of a formal security definition is to *see an attack that breaks the security definition and try to interpret its real-world implications*: it is hard to fully appreciate that the standard UC PAKE security notion covers

[12] [BCP+23] uses a version where the protocol messages c and c' are included in the final hash, and reduces to DDH instead of CDH.

forward secrecy, or even what forward secrecy means, without seeing the attack in Section 2.2. For the second argument, we note that PAKE is not merely a topic of practical relevance: boosting the entropy of the shared string — in a way that is resilient to man-in-the-middle adversaries — is of great theoretical significance, and it might not even be a priori clear whether it was feasible (let alone done so efficiently).

But the rebuttal above is not our main point. Rather, we believe the central question here is: *How important are security proofs?* One defining characteristic of "modern cryptography" is the central role of security proofs, which are deemed necessary for any reasonable public-key protocol. But then it must follow that (non-trivial) flaws in security proofs need to be treated seriously: *even if there is no attack whatsoever, a flawed proof should mean that the protocol is not to be trusted — until someone gives a correct proof.* In our opinion, how serious a flaw is should depend on not only whether the protocol itself is secure or not, but also how hard it is to fix the *proof*.

As we have seen, the crux of the EKE security proof is to deal with the adversary in Figure 4, which passes the first message but modifies the second (hybrid 3 in our proof, which relies on 1-ODH) — which is overlooked in all of the works above. As such, all these proofs are broken in a fundamental sense, and repairing them turns out to be highly non-trivial: it requires careful analysis to even identify this "most problematic" case, let alone discover the tightness gap while reducing to CDH. Given that EKE is the first PAKE protocol and arguably the most thoroughly studied, we believe a correct and clearly written security proof has been long overdue.

Let us conclude by returning to the quote at the beginning. Many students find it hard to grasp even the basic concepts in mathematical analysis and to write even a simple proof, not because the intuition is unclear, but rather because of the complexity of the formal definitions and their disconnection with the intuition.[13] After centuries of struggle, the mathematical community finally found out that the more intuitive "infinitesimal" approach could form a rigorous basis for analysis (nonstandard analysis) — a contribution to mathematics education, a significant result in mathematical logic, and a useful tool in analysis itself. Similarly, security definitions for cryptographic protocols such as PAKE — especially those that guarantee security under composition — could be prohibitive to beginners, primarily because the formal definitions are extraordinarily complex and look distant from the intuition.[14] (The difference is also obvious though: the "infinitesimal" approach to analysis had been used for centuries and produced innumerable correct results before the advent of Weierstrass's (ε, δ)-notion; whereas no alternative paradigm for security definitions has been even remotely as successful as standard ones such as UC.) Is there a more intuitive, potentially radically different, yet equally rigorous way to define the security of protocols? We hope history will give a positive answer, but for now we have to stick to the UC (and game-based) security definition(s) for PAKE...[15]

[13] Quoting [Blu19]: *Researchers have examined student difficulties coming from [the (ε, δ)-definition's] multiple nested quantifiers as well as its great distance from the less formal notions of limit with which students typically enter its study.*

[14] Coincidentally, both definitions have three quantifiers (and in the same order): "$\forall\varepsilon\exists\delta\forall x$" in the (ε, δ)-definition, versus "$\forall\mathcal{A}\exists\mathcal{S}\forall\mathcal{Z}$" in UC. Of course, if you expand the definition of indistinguishability in UC, there are three more quantifiers waiting for you.

[15] We should add that we disagree with the opinion that security definitions have become too complicated these days and should be (somewhat) disregarded. They are like the (ε, δ)-notion in cryptography, i.e., the *only way* to formally define an important concept (before someone finds a simpler definition that is equivalent) — no matter how nonintuitive and inconvenient. If mathematicians can deal with the (ε, δ)-notion, why can't cryptographers deal with complicated security definitions? Also, we believe part of the issue is that there are a lot of UC functionalities without a detailed and beginner-friendly explanation, which this tutorial attempts to remedy a bit.

5 Exercises

1. In the standard UC PAKE functionality [CHK+05, Figure 2], there is a role field in the NewSession command, which does not appear in our Figure 2. What is its purpose, and is it necessary?
   Hint: It has little to do with the security of the protocol.

2. We have seen that if the session key is $H(g^{xy})$, then the reduction to CDH needs to lose a factor of $\Theta(q^2)$. Would the reduction become tighter if the session key was $H(g^x, g^y, g^{xy})$ (note that the two parties can recover g^x and g^y upon IC decryption)? What if it was $H(c, c', g^{xy})$? What about $H(\mathsf{pw}, c, c', g^{xy})$?
   Hint: A partial answer can be found in [JRX25, Remark A.1].

3. Complete the omitted HDH reductions in hybrids 5, 6, 8, 10. (As discussed above, you should not trust me saying these cases are "similar" to hybrid 2, and should work out the reductions in detail. In particular, the reduction in hybrid 5 needs some thought: the high-level argument only mentions the P' side, but the reduction needs to simulate the P side as well. Also, in hybrid 10 think about why a single reduction suffices for all three sub-cases.)

4. In the simulator in Figure 7, the case that the adversary modifies the first message (using a correct password guess) but passes the second is handled separately in step 3(b). What if we remove this case and treat it as in step 3(c)? Will the simulator fail?

5. In our security proof, the cases that $\mathsf{pw} \neq \mathsf{pw}' \wedge c^* = c$ and $\mathsf{pw} = \mathsf{pw}' \wedge c^* = c$ are handled separately (in resp. hybrids 2 and 3): the former case can be reduced to HDH, whereas we have to rely on 1-ODH in the latter case. Can we combine these two cases and do a single reduction to 1-ODH in the $c^* = c$ case? If so, how would the reduction go? If not, where does it break down?

6. In hybrid 3, we reduce to 1-ODH, which in turn can be reduced to CDH (with a $\Theta(q^2)$ loss). If we do not define 1-ODH and instead reduce hybrid 3 to CDH directly, how should that reduction go? (This will probably make the argument less modular and less clear. Nevertheless, it is a good exercise to think through this "lower-level" reduction.)

7. The *implicit-only* PAKE functionality [DHP+18, Fig.8] is the same as the standard one in Figure 2, except that the ideal adversary $\mathcal{A}^*$ does not receive "correct/wrong guess" on TestPwd. Does EKE realize this functionality? (Note that the current simulator relies on the "correct/wrong guess" response on TestPwd in steps 2(b)(ii), 3(b), and 3(c)(ii).)

References

[ABR01] Michel Abdalla, Mihir Bellare, and Phillip Rogaway. The oracle Diffie–Hellman assumptions and an analysis of DHIES. In *CT-RSA 2001*, pages 143–158, 2001.

[BCP+23] Hugo Beguinet, Céline Chevalier, David Pointcheval, Thomas Ricosset, and Mélissa Rossi. GeT a CAKE: Generic transformations from key encaspulation mechanisms to password authenticated key exchanges. In *ACNS 2023*, pages 516–538, 2023.

[Blu19] Ben Blum-Smith. Some thoughts about epsilon and delta, 2019. https://blogs.ams.org/matheducation/2019/08/19/some-thoughts-about-epsilon-and-delta.

[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.

[Cho09] Timothy Y. Chow. A beginner's guide to forcing. In Joseph A. Gallian, Timothy Y. Chow, and Daniel C. Isaksen, editors, *Communicating Mathematics*. 2009.

[Cry20] Crypto Forum Research Group. PAKE selection, 2020. https://github.com/cfrg/pake-selection.

[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.

[FGJ23] Bruno Freitas Dos Santos, Yanqi Gu, and Stanislaw Jarecki. Randomized half-ideal cipher on groups with applications to UC (a)PAKE. In *EUROCRYPT 2023*, pages 128–156, 2023.

[Gol12] Oded Goldreich. On errors (part II), 2012. https://www.wisdom.weizmann.ac.il/~oded/etc.html#op18.

[Gra83] Judith V. Grabiner. Who gave you the epsilon? Cauchy and the origins of rigorous calculus. *The American Mathematical Monthly*, 90(3):185–194, 1983.

[Jar23] Stanislaw Jarecki. Randomized half-ideal cipher on groups with applications to UC (a)PAKE. https://www.youtube.com/watch?v=GL4m7StDsPg, 2023. Talk at EUROCRYPT 2023.

[JKSS12] Tibor Jager, Florian Kohlar, Sven Schäge, and Jörg Schwenk. On the security of TLS-DHE in the standard model. In *CRYPTO 2012*, pages 273–293, 2012.

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

[MRR20] Ian McQuoid, Mike Rosulek, and Lawrence Roy. Minimal symmetric PAKE and 1-out-of-N OT from programmable-once public functions. In *CCS 2020*, pages 425–442, 2020.

[pg117] pg1989. Answer to "why would someone prove security in the real-ideal model instead of universal composibility framework?", 2017. https://crypto.stackexchange.com/a/46941.

[PZ23] Jiaxin Pan and Runzhi Zeng. A generic construction of tightly secure password-based authenticated key exchange. In *ASIACRYPT 2023*, pages 143–175, 2023.

[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.