The Most Efficient Protocol for PAKE:
What Exact Stuff Do You Need to Hash at the End? *

Jiayu Xu†

Abstract

A Password-Authenticated Key Exchange (PAKE) protocol allows two parties to jointly establish a cryptographic session key, in the "password-only" setting where the only information shared in advance is a low-entropy password. In recent years, the One-encryption EKE with 2-round Feistel cipher (OEKE-2F) protocol, a compiler from Key Encapsulation Mechanism (KEM) to PAKE, has received much attention, for the following reasons: (1) When instantiated with the Diffie–Hellman KEM, it is the most computationally efficient PAKE protocol to date that is secure in the Universal Composability (UC) framework; and (2) When instantiated with a post-quantum KEM, it provides a generic way to construct efficient PAKE protocols based on post-quantum assumptions.

Unfortunately, the community cannot agree upon what the OEKE-2F protocol exactly is: part of the second protocol message is an RO hash of the KEM key, together with any number of the following:

- The password,
- The KEM public key,
- The first protocol message, and
- The KEM ciphertext.

This yields 16 potential variants of OEKE-2F; only two of them have been studied in the literature, and their pros and cons are poorly understood.

In this work, we present a comprehensive analysis of *all 16 variants* of OEKE-2F, proving the UC-security of each of them. The general takeaway is that the "hash everything" version requires the fewest security properties of the underlying KEM scheme, and the more items we remove from the hash, the more security requirements the KEM scheme has to satisfy — although all of the additional KEM properties are still mild. We pinpoint the exact KEM properties each version of OEKE-2F needs, and thoroughly explain the rationales.

The significance of this work lies in that it helps the community converge upon the "right" version of OEKE-2F, and perhaps also in that this is the first paper by the author that is over 100 pages.

*The `.tex` code of this paper is available at https://github.com/jiayux123/oeke. Some minor LaTeX hacks, such as lines 18–25, 36–38, 1078, 1121–1122, 1975–1977 and 2074, might be useful to beginners.

†Oregon State University, xujiay@oregonstate.edu

Contents

1 Introduction **4**
- 1.1 Motivating Example: OEKE-2F with Diffie–Hellman KEM . . . 10
- 1.2 Our Contributions . . . 12
  - 1.2.1 Reanalysis of the "Hash Everything" Version . . . 12
  - 1.2.2 Analyses of the Other 15 Variants . . . 14
- 1.3 How to Read This Paper . . . 15

2 Preliminaries and Technical Overview **16**
- 2.1 KEM Properties Used in the "Hash Everything" Version . . . 17
- 2.2 Additional KEM Properties Used in Other Versions . . . 20
- 2.3 Example: Diffie–Hellman KEM . . . 26
- 2.4 The UC PAKE Functionality . . . 28

3 UC Description of the OEKE-2F Protocol **29**

4 Security Analysis of the "Hash Everything" Version **31**
- 4.1 The Simulator . . . 31
  - 4.1.1 Simulation Strategy . . . 31
  - 4.1.2 Formal Description of the Simulator . . . 33
- 4.2 Game-Hop Argument . . . 35

5 Security Analyses of Other Versions (I): Including (s, T) in the Hash **57**
- 5.1 What Will Go Wrong If You Don't Hash Everything? . . . 57
- 5.2 The "Hash pw, pk and (s, T)" Version . . . 59
- 5.3 The "Hash pw and (s, T)" Version . . . 60
- 5.4 The "Hash pk and (s, T)" Version . . . 61
  - 5.4.1 Changes to the Simulator . . . 61
  - 5.4.2 Changes in Game Hops . . . 62
- 5.5 The "Only Hash (s, T)" Version . . . 64
  - 5.5.1 Changes to the Simulator . . . 64
  - 5.5.2 Changes in Game Hops . . . 65

6 Future Directions **68**

References **70**

A Comparison Between Our Simulator and the One in [ABJ25] **74**

B Security Analyses of Other Versions (II): Removing (s, T) from the Hash **74**
- B.1 The "Hash pw and pk" Version . . . 74
  - B.1.1 Changes to the Simulator . . . 75
  - B.1.2 Changes in Game Hops . . . 77
- B.2 The "Hash pw" Version . . . 83

B.2.1 Changes to the Simulator 84
B.2.2 Changes in Game Hops 85
B.3 The "Hash pk" Version 90
B.3.1 Changes to the Simulator 90
B.3.2 Changes in Game Hops 92
B.4 The "Hash Nothing But K" Version 96
B.4.1 Changes to the Simulator 96
B.4.2 Changes in Game Hops 97

C Comparison with [JRX25] 101

D Deferred Proofs from Section 2 103
D.1 Proof of Lemma 1 103
D.2 Proof of Lemma 2 106
D.3 Proof of Lemma 3 107

1 Introduction

Ever since the seminal work by Bellovin and Merritt in 1992 [BM92], *Password-Authenticated Key Exchange (PAKE)* has been an enduring topic in cryptography. Such a protocol allows two parties to jointly establish a cryptographically strong session key in the so-called "password-only" setting, where the only information shared in advance is a low-entropy password; crucially, it must be secure in the presence of an MitM adversary that can modify protocol messages in an arbitrary manner. PAKE has received much attention from both theoretical and applied communities: theoretically speaking, it achieves something that might appear impossible — bootstrapping the entropy of a shared secret in such a way that is secure against the strongest network adversary — and recent years have witnessed some efforts on the standardization and deployment of PAKE protocols in practice [Cry20].

Encrypted Key Exchange (EKE). Apart from introducing PAKE the concept, [BM92] also proposed a highly influential PAKE protocol called *Encrypted Key Exchange (EKE)*. This is a general compiler from any Key Encapsulation Mechanism (KEM) scheme[1] to a PAKE, and its idea is simple: the two protocol parties (symmetrically) encrypt the KEM public key and ciphertext under their password, respectively, and send the results as protocol messages; holding the same password allows them to decrypt the protocol messages and complete the underlying KEM scheme, which yields the shared KEM key; finally, the two parties output an RO hash of the KEM key as their PAKE session key.

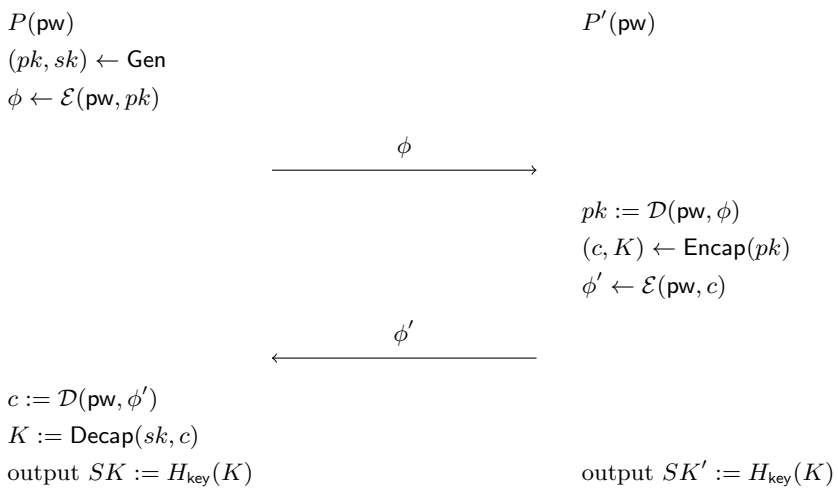

Figure 1: The EKE protocol with KEM scheme $(\mathsf{Gen}, \mathsf{Encap}, \mathsf{Decap})$ and symmetric encryption $(\mathcal{E}, \mathcal{D})$

[1] EKE is often described using the terms of (unauthenticated) key exchange protocols, but a 2-message-flow key exchange is equivalent to a KEM (see, e.g., [SGJ23, Section 2.2]).

EKE is almost as efficient as the underlying KEM scheme, with one exception: both parties need to perform an encryption of the KEM public key/ciphertext and a decryption of the protocol message. For a long time, this encryption mechanism was generally modeled as an Ideal Cipher (IC), which allows the security proof for EKE to go through [BPR00]. While IC over bitstrings can be implemented using (say) AES, EKE requires an IC *over the KEM public key/ciphertext space*, e.g., a cyclic group if we use the Diffie–Hellman KEM, and it is unclear how to instantiate such an IC efficiently. A standard result [DS16] shows that an IC is indifferentiable from an 8-round Feistel cipher, assuming that the underlying round function is an RO; however, applying this IC construction to EKE would mean an overhead of 8 hash-to-group operations for each protocol party (4 in encryption and 4 in decryption), rendering the efficiency of EKE not ideal.

One-encryption EKE (OEKE). Over the years, there have been multiple efforts aimed at improving the efficiency of EKE. The first among them is the variant called *One-encryption EKE (OEKE)*, proposed by Bresson, Chevassut and Pointcheval in 2003 [BCP03]. Here, the first protocol message is exactly the same as in EKE, but second protocol message is replaced by the "raw" KEM ciphertext together with an authenticator defined as an RO hash of the KEM key. In this way, we remove the evaluation of IC in one direction.

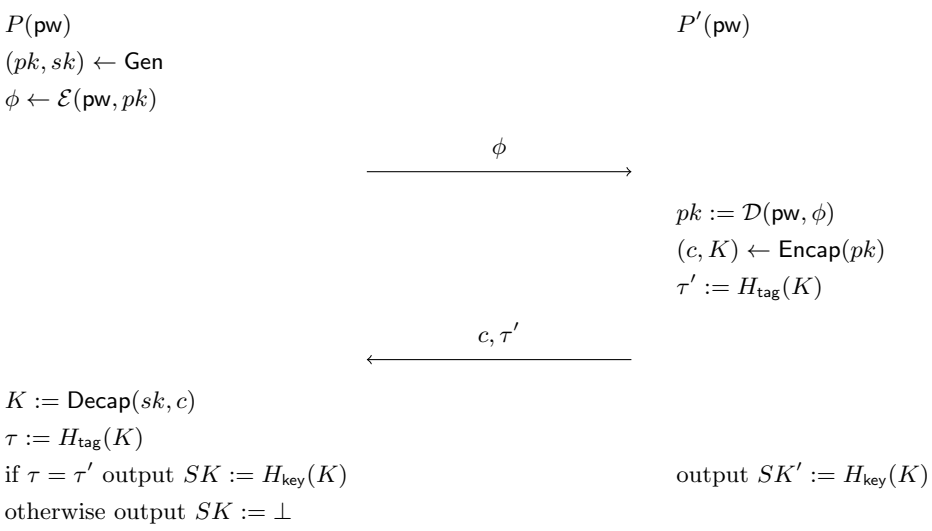

Figure 2: The OEKE protocol with KEM scheme $(\mathsf{Gen}, \mathsf{Encap}, \mathsf{Decap})$ and symmetric encryption $(\mathcal{E}, \mathcal{D})$

Apart from being computationally more efficient than EKE, OEKE also achieves one-side *explicit authentication*: the initiator (P in Figure 2) can check

if its counterparty holds the correct password by computing its own authenticator and comparing it with the incoming message; if the check fails, it can explicitly reject.[2]

Using 2-round Feistel cipher (2F) in place of IC. In 2020, an orthogonal optimization of EKE was proposed by McQuoid, Rosulek and Roy [MRR20]. Their trick is simple: While a 2-round Feistel cipher (2F) is differentiable from an IC, it nevertheless can be used in place of IC in EKE. In more detail, they use the following procedure

$$
\begin{array}{l}
\underline{\mathsf{2F}(\mathsf{pw}, pk)}: \\
r \leftarrow \{0,1\}^{3n}, \\
R := H_1(\mathsf{pw}, r), \qquad\qquad T := pk \cdot R, \\
t := H_2(\mathsf{pw}, T), \qquad\qquad s := r \oplus t, \\
\text{output } \phi = (s, T)
\end{array}
$$

instead of IC encryption $\mathcal{E}(\mathsf{pw}, pk)$, and the following procedure

$$
\begin{array}{l}
\underline{\mathsf{2F}^{-1}(\mathsf{pw}, \phi = (s, T))}: \\
t := H_2(\mathsf{pw}, T), \qquad\qquad r := s \oplus t, \\
R := H_1(\mathsf{pw}, r), \qquad\qquad pk := T/R, \\
\text{output } pk
\end{array}
$$

instead of IC decryption $\mathcal{D}(\mathsf{pw}, \phi)$. This mechanism can be viewed as a weaker, randomized variant of IC, which [MRR20] calls Programmable-Once Public Function (POPF). [MRR20] showed the security of EKE with 2F,[3] whose computational overhead is 2 hash-to-group operations on each side (one evaluation of H_1 in $\mathsf{2F}$ and one in $\mathsf{2F}^{-1}$; note that H_2 is onto bitstrings) as opposed to 8.

OEKE-2F. Interestingly, [MRR20] did not mention OEKE, and an immediate future direction was to replace the (single) IC in OEKE with 2F. This was the subject of two works in 2025, one by Januzelli, Roy and Xu [JRX25] and the other by Arriaga, Barbosa and Jarecki [ABJ25]. While the two works drastically differ in the presentation of security proofs, both showed that OEKE with 2F (henceforth OEKE-2F) is secure in the Universal Composability (UC) framework [Can01], the current security standard for PAKE. The computational overhead of OEKE-2F is only one hash-to-group operation on each side; indeed, when instantiated with the Diffie–Hellman KEM, this is the most computationally efficient PAKE protocol that has ever been proven UC-secure under the

[2] It is unclear whether this should be called "initiator-side explicit authentication" or "respondent-side explicit authentication": It is the initiator who may explicitly reject, but what's checked is whether the respondent holds the correct password. Below we use the term "initiator-side explicit authentication".

[3] The security proof in [MRR20] is flawed, which was later fixed in [JRX25].

standard notion for UC PAKE.[4]

Apart from its computational efficiency, OEKE-2F has received much attention for two other reasons:

- Since OEKE-2F is a generic compiler from *any* KEM scheme (with certain reasonable security properties) to PAKE, plugging in a post-quantum KEM scheme immediately yields a PAKE protocol secure under post-quantum assumptions. In particular, it is a key building block of the CPaceOQUAKE hybrid PAKE protocol, currently under consideration for standardization [VJW26].
- A recent variant of OEKE-2F called Tempo [ABJ26] is resilient to timing attacks, while mostly retaining the efficiency of OEKE-2F.

Remark 1. We caution that "PAKE secure under post-quantum assumptions" and "post-quantum PAKE" are two different concepts: the latter additionally requires a security proof in the Quantum-accessible ROM (QROM), and such a proof for (O)EKE appears difficult [HHKR25]. ([VJW26] calls their protocol "post-quantum" without an analysis in the QROM, which is, in our opinion, misleading.) Nevertheless, it is fair to say that a security analysis under post-quantum assumptions could serve as the first step of a post-quantum analysis.

	with IC	with 2F
EKE	[BM92, BPR00] 8 hash-to-group	[MRR20] 2 hash-to-group
OEKE	[BCP03] 4 hash-to-group	[JRX25, ABJ25] 1 hash-to-group

Table 1: Optimizations of EKE, and their computational overheads over the KEM scheme

What exact stuff do you need to hash at the end? The above appears to be the happy ending of the whole story, and the reader might wonder why I wrote this (super long) paper at all. The reason is simple: *The OEKE-2F protocols analyzed in [JRX25] and [ABJ25] are different!* Looking at their respective protocol descriptions ([JRX25, Figure 19] and [ABJ25, Fig.4]), one can realize that the authenticator τ' and the session key SK are derived differently in these two works: in [JRX25] they are $H_{\mathsf{tag}}(K)$ and $H_{\mathsf{key}}(K)$, i.e., exactly as in Figure 2, whereas in [ABJ25] they are $H_{\mathsf{tag}}(\mathsf{pw}, pk, s, T, c, K)$ and $H_{\mathsf{key}}(\mathsf{pw}, pk, s, T, c, K)$, i.e., the hashes of the KEM key K together with

[4] A few other PAKE protocols, such as AuthA [BCP04], SPAKE2 [AP05] and CPace [HL19], have computational costs comparable to that of OEKE-2F (a concrete comparison is difficult, since the tradeoff is between hash-to-group operations and multiexponentiations in the group, and thus depends on the implementation details). However, none of these protocols is likely UC-secure under the standard notion (see [ABB+20, Section 1] for a discussion). SPAKE2 and CPace do not achieve one-side explicit authentication. All these protocols use a Diffie–Hellman group and are not generic KEM-to-PAKE compilers.

- The password pw,
- The KEM public key pk,
- The first protocol message (i.e., the 2F ciphertext) $\phi = (s, T)$, and
- The KEM ciphertext c.

The two protocols analyzed in [JRX25, ABJ25] are two extremes on a spectrum of OEKE variants: For each of the four items above, one could choose to include or exclude it in the ROs, resulting in $2^4 = 16$ versions of OEKE. By examining the literature on OEKE, an immediate impression is that **people simply cannot agree upon what the OEKE protocol exactly is**. Below we list all existing security analyses of OEKE, and what exact items are hashed.

paper	protocol name	items hashed	encryption mechanism	security notion
[BCP03]	OEKE	pk, c	IC	game-based
[SGJ23]	EKE-KEM	pk	half IC	UC
[BCP+23]	OCAKE	pw, ϕ, c	IC	UC
[AHH+24]	OCAKE	pw, pk, ϕ, c	IC	game-based
[JRX25]	OEKE		2F	UC
[ABJ25]	NoIC	pw, pk, ϕ, c	2F	UC

Table 2: Existing works on OEKE. The KEM key K is always hashed and omitted from the table. We omit the machine-aided analysis in [Bla12] and the post-quantum analysis in [HHKR25], as they are not the focus of this work

This is a confusing, and frankly embarrassing, situation: everyone gives their own OEKE security analysis with some items included in the hash, which is a seemingly arbitrary choice by the authors; it has never been studied — or even sufficiently explained in an informal manner — what exact items need to be included, and *why*. In particular, for the state-of-the-art OEKE-2F, only the "hash everything" and "hash nothing but K" versions have been analyzed, and it is unclear whether, or why, those items need to be included in the hash. (Another confusing point is that everyone has their own name for OEKE.)

Why not simply hash everything? An immediate question is whether this topic matters at all — why not simply hash everything like what [ABJ25] did? This question, in fact, has been partially answered in [ABJ25, Section 5], and below we summarize our understanding:

- Since the initiator's algorithm has two stages, of course *some* state has to be memorized in between. From Figure 2 it is clear that the KEM secret key sk has to be part of the state. However, the more items we include in the hash, the more the initiator needs to memorize. Therefore, hashing fewer items is beneficial in the setting where the initiator's memory is extremely limited.

- Including the password pw in the hash is especially problematic, since if the initiator's memory is stolen between its two stages, then pw is immediately leaked. In other words, removing pw from the hash helps defend against side-channel attacks better. The same goes for the KEM public key pk (note that pk is supposed to be secret in OEKE: if pk is leaked then an eavesdropper can run an offline dictionary attack by trying to decrypt $\mathsf{2F}^{-1}(\mathsf{pw}^*, (s, T))$ for multiple candidate passwords pw^* and check which one yields pk). In contrast, the first protocol message ϕ and the KEM ciphertext c are supposed to be seen by the adversary anyway, so they are not a concern in this regard.

- Having a longer hash input means more calls to the underlying compression function while computing the hash, so hashing fewer items improves the computational efficiency (admittedly the bottleneck of computational cost is the KEM and the hash-to-group operations, and the optimization here is likely minor).

These reasons aside, perhaps the primary motivation of this work is that

> *The current state of the art, where the community cannot converge upon a "right" version of OEKE and everyone does an ad hoc analysis, is just sad, especially given that (1) OEKE has been around for over 20 years, and (2) it is one of the most significant PAKE protocols as explained earlier.*

Furthermore, we believe the topic we study is of theoretical interest: The requirements on the underlying KEM scheme depend on the items included in the hash, and it is very interesting to see what exact additional KEM properties are needed when we remove certain items from the hash. Also, the security proofs require careful considerations of the order of games and how the reductions can go through.

Why is this paper so long? A common complaint about PAKE papers is that they tend to be long and hard to understand, especially if they prove UC-security. I claim that part of it is due to the nature of PAKE and/or UC, and cannot be circumvented. The main reason is that there are a number of factors to consider, including

- Whether the MitM adversary forwards or modifies a protocol message;

- If the adversary modifies a protocol message, whether the adversarial message contains a correct password guess, an incorrect password guess, or no password guess; and

- Whether the two honest parties' passwords match.

Combined, they yield a large amount of cases that need to be covered in the security proof. For example, there has to be (at least) one game showing that if the adversary sends a message that contains an incorrect password guess, then

the receiving party's session key is indistinguishable from random; another game has to show that if the adversary is passive (i.e., forwards both protocol messages without modification) and the two honest parties' passwords are different, then the two parties output independently random session keys.[5]

Still, there are other reasons specific to this paper. One is, of course, that there are too many OEKE variants to analyze. The reason why the introduction and preliminaries (Section 2) sections of this paper are unusually long is the following: I wish to explain this work to such an extent that by reading Sections 1 and 2 alone, one could gain an intermediate-level understanding of the technical details — including why certain KEM properties are needed in certain OEKE variants — and you probably don't need to read the remaining sections. In this way, the "effective length" of this paper is roughly 25 pages.

1.1 Motivating Example: OEKE-2F with Diffie–Hellman KEM

One might ask why removing certain items from the hash requires the underlying KEM scheme to have some additional security properties. To see the subtleties, let's work out a concrete example: OEKE-2F with the Diffie–Hellman KEM scheme. We show the "hash nothing but K" version below.

$P(\mathsf{pw})$		$P'(\mathsf{pw})$
$x \leftarrow \mathbb{Z}_q, X := g^x$		
$(s, T) \leftarrow \mathsf{2F}(\mathsf{pw}, X)$		
	$\xrightarrow{s, T}$	
		$X := \mathsf{2F}^{-1}(\mathsf{pw}, (s, T))$
		$y \leftarrow \mathbb{Z}_q, Y := g^y$
		$K := X^y$
		$\tau' := H_{\mathsf{tag}}(K)$
	$\xleftarrow{Y, \tau'}$	
$K := Y^x$		
$\tau := H_{\mathsf{tag}}(K)$		
if $\tau = \tau'$ output $SK := H_{\mathsf{key}}(K)$		output $SK' := H_{\mathsf{key}}(K)$
otherwise output $SK := \perp$		

Figure 3: The OEKE-2F protocol with Diffie–Hellman KEM

One does not need to understand the UC PAKE functionality — or *anything*

[5] This assumes implicit authentication; in explicit authentication the party(ies) should output the rejection symbol instead of a random session key.

about UC — to see that this protocol is completely insecure. The adversary $\mathcal{A}$ disregards P', and sends $(Y = 1, \tau' = H_{\mathsf{tag}}(1))$ to P; this will cause P's check to pass, and P will compute its session key as $H_{\mathsf{key}}(Y^x) = H_{\mathsf{key}}(1)$. In other words, $\mathcal{A}$ can predict P's session key without knowing P's password — which shouldn't be allowed by any reasonable security notion of PAKE.

There are three ways to fix this:

- If we include pw in H_{tag} inputs, i.e., define τ' as $H_{\mathsf{tag}}(\mathsf{pw}, K)$, then this immediately thwarts the above attack: coming up with a valid τ' requires $\mathcal{A}$ to know pw.
- Alternatively, we could also include the KEM public key X in H_{tag} inputs, i.e., define τ' as $H_{\mathsf{tag}}(X, K)$. Then coming up with a valid τ' requires $\mathcal{A}$ to know X, which in turn requires $\mathcal{A}$ to know pw since the only way for $\mathcal{A}$ to learn X is to decrypt (s, T) under pw.
- Perhaps the most straightforward fix is to disallow the identity group element 1 as a valid protocol message, i.e., P should immediately reject upon receiving $Y = 1$.

If we only care about OEKE-2F with Diffie–Hellman KEM, then we are basically done here (it is common practice to exclude 1 in Diffie–Hellman anyway). However, keep in mind that we want to analyze a *generic compiler* from KEM to PAKE, and on this level, the lesson is that

> *If we remove* ***both*** pw *and pk from* H_{tag} *inputs, then the underlying KEM scheme needs to satisfy an additional security property.*

But what exactly is the additional property? For the specific Diffie–Hellman KEM scheme, this is clear: exclude 1. However, it is non-trivial to figure out the required KEM property *in the general case*. Looking ahead, what's needed is [click to reveal spoiler]

.

The above is only a simple example of the tradeoff between "what's included in the hash" and "what properties the KEM needs to satisfy"; we will see multiple other examples below: removing c from the hash requires an additional KEM property, removing *both* pk and (s, T) from the hash requires another additional KEM property, and so on.

Remark 2. The above assumes that the session key SK is defined as $H_{\mathsf{key}}(K)$. If $SK = H_{\mathsf{key}}(\mathsf{pw}, K)$ and $\tau' = H_{\mathsf{tag}}(K)$, then the attack results in P outputting a session key uniformly random in $\mathcal{A}$'s view (since $\mathcal{A}$ does not know pw). In general the attack can be thwarted as long as *one of* H_{key} and H_{tag} has pw or pk as input, although including the input in H_{tag} is "better" since it allows for

explicit authentication. For simplicity, below we always assume $SK = H_{\mathsf{key}}(K)$, i.e., H_{key} takes the "bare" input K, and only consider what's included in H_{tag}.

Remark 3. The Diffie–Hellman example in this section is taken from [JRX25, Section 3.1]; in particular, [JRX25, Remark 3.3] points out that the issue can be fixed by including either pw or pk.

1.2 Our Contributions

As mentioned above, any number of $\mathsf{pw}, pk, (s, T), c$ can be included in H_{tag} inputs, resulting in 16 variants of OEKE-2F; among them, only the "hash nothing but K" and "hash everything" versions have been previously studied (in [JRX25] and [ABJ25], respectively). This work provides a comprehensive and thorough study of *all 16 variants*, revealing the requirements on the underlying KEM scheme for each of them. Below we summarize our contributions.

1.2.1 Reanalysis of the "Hash Everything" Version

In Section 4 we analyze the "hash everything" version of OEKE-2F, where the authenticator τ' is defined as $H_{\mathsf{tag}}(\mathsf{pw}, pk, s, T, c, K)$. We choose this protocol as our first target because it is the "easiest" one: it requires the fewest properties of the KEM scheme, and it has already been proven UC-secure in [ABJ25].

Of course, the immediate question is why we analyze this protocol again. The answer is that we make multiple improvements over [ABJ25], which we elaborate below.

Improvement #1: modeling explicit authentication. As we have discussed, OEKE has the feature of one-side explicit authentication (1EA): the initiator can tell whether authentication succeeds or not, and output a real session key or the rejection symbol $\perp$ accordingly. This is, unfortunately, not modeled in [ABJ25] (nor in [JRX25]): if authentication fails, they let the initiator output a uniformly random string as the session key instead of $\perp$. The reason is that the standard UC PAKE functionality is implicit-authentication-only, i.e., it does not allow a protocol party to output $\perp$; thus, the protocol in [ABJ25] has to do away with 1EA in order to fit the UC PAKE functionality.

In this work, we propose a UC functionality for PAKE-1EA, and show that the original OEKE-2F protocol — where the initiator outputs a real session key or $\perp$ — realizes it. In this way, we prove for the first time in the literature that (any variant of) OEKE satisfies the 1EA property.[6] Our PAKE-1EA functionality only involves a minor modification to the standard PAKE functionality.

Improvement #2: weaker KEM properties. The last paragraph of [ABJ25], titled "minimum requirements for the KEM", claims that the "hash everything"

[6] [BCP+23] also gives a PAKE-1EA functionality, but their functionality is problematic and cannot be realized by OEKE (see Remark 8 below).

version of OEKE-2F is UC-secure assuming the underlying KEM scheme has the following three properties:

- Pseudorandom public keys (UNI-PK),
- One-wayness under one plaintext-checking attack (OW-1PCA), and
- Anonymity under one plaintext-checking attack (ANO-1PCA).

Close scrutiny shows that both of their definitions of OW-1PCA and ANO-1PCA are too strong, and weaker properties (which we still call OW-1PCA and ANO-1PCA) suffice for OEKE-2F security. Roughly speaking, in OW-1PCA we could disallow the adversary from querying the challenge ciphertext to the plaintext-checking oracle, and in ANO-1PCA we could give the adversary only one public key instead of both. We refer to Section 2.1 for a formal discussion.

Improvement #3: fixing flaws. We believe the security proof in [ABJ25] is mostly correct; in particular, it considers a number of subtleties carefully. However, in the following two cases:

- The two parties' passwords are different, and the adversary is passive; and
- The two parties' passwords are equal, and the adversary is passive except that it modifies the authenticator τ',

the proof in [ABJ25] fails to argue that the UC environment's views in the real world and the ideal world are indistinguishable. We refer to Games 12 and 14 in Section 4.2 for a detailed explanation.

Improvement #4: more detailed presentation of the security proof. Perhaps our most significant improvement over [ABJ25] lies in the presentation of the security proof. The most complicated and subtle step of the proof is the last game (our Game 17 in Section 4.2, their games G_{18} and G_{19} in Section 5); the argument in [ABJ25] has a "reduction within reduction", i.e., the reduction simulates the game in a manner that is indistinguishable from, but not identical to, the real game, and a second "inner" reduction is needed to show this indistinguishability. This complicated proof strategy makes their argument hard to follow, and is in fact unnecessary: we show how to make a cleaner argument by defining a separate, tailored KEM property, which is implied by standard properties and which we reduce to in the game.

We list some other improvements on the presentation below:

- We drastically simplify the description of the UC simulator by removing unnecessary records.
- A few other reductions, including the ones to OW-1PCA and ANO-1PCA, are also quite complicated, and we give detailed descriptions instead of the sketches in [ABJ25]. (From our descriptions of these reductions, it is immediately clear why the OW-1PCA and ANO-1PCA properties in [ABJ25] are stronger than necessary.)

- Our Game 9 in Section 4.2, which is equivalent to game G_{13} in [ABJ25, Section 5], essentially says that the simulator can simulate the P-to-P' message (s, T) at random and then program the ROs later if necessary. Our statistical argument why this game hop goes through is much cleaner than theirs.

1.2.2 Analyses of the Other 15 Variants

Once we fully work out the "basic" security proof for the "hash everything" version of OEKE, in Section 5 and Appendix B we turn to other variants of the protocol. Since the only difference lies in how the P'-to-P authenticator τ' is defined, a large portion of the argument — e.g., password extraction from the adversarial P-to-P' message, and P''s reaction upon receiving this message — is irrelevant to the OEKE variant we are analyzing and can be reused. However, as seen in Section 1.1, removing certain items from the hash puts more requirements on the KEM scheme, and figuring out these additional properties is highly non-trivial. Below we list the KEM properties needed for all 16 OEKE variants.

items hashed				KEM properties needed	analyzed in...
pw	pk	(s, T)	c		
●	●	●	●		Section 4
●	●	●		CBIND	Section 5.2
●		●		CBIND, UNPRE	Section 5.3
●		●	●	UNPRE	Section 5.3
	●	●		CBIND	Section 5.4
	●	●	●		Section 5.4
		●		CBIND, KBIND, CR	Section 5.5
		●	●	CR	Section 5.5
●	●			CBIND	Appendix B.1
●	●		●		Appendix B.1
●				CBIND, UNPRE, KBIND	Appendix B.2
●			●	UNPRE, KBIND	Appendix B.2
	●			CBIND	Appendix B.3
	●		●		Appendix B.3
				CBIND, KBIND, CR	Appendix B.4
			●	KBIND, CR	Appendix B.4
K is always hashed, which is omitted above UNI-PK, OW-1PCA and ANO-1PCA properties are always required UNPRE is implied by CR (Lemma 3)					

Table 3: List of OEKE variants. The KEM properties are not necessarily minimal (see Section 6)

We will define the various additional KEM properties in Section 2.2, but at a high level, they are different types of binding properties. For example, if the

KEM ciphertext c is included in the hash, then modifying c but forwarding τ' will immediately cause P to reject; however, if c is removed from the hash, then we need some additional "ciphertext binding" property to prevent the adversary from doing this. We show the full names of the KEM properties below.

name	full name	needed if ... is excluded from the hash	defined in...
UNI-PK	pseudorandom public keys	(needed in all variants)	Definition 1
OW-1PCA	one-wayness under one plaintext-checking attack	(needed in all variants)	Definition 3
ANO-1PCA	anonymity under one plaintext-checking attack	(needed in all variants)	Definition 5
CBIND	ciphertext binding	c	Definition 7
UNPRE	unpredictability	pk	Definition 8
KBIND	key binding	pw, pk, c; or $pk, (s, T)$	Definition 9
CR	collision resistance	pw, pk	Definition 10

Table 4: KEM properties needed for OEKE variants

The information above could be overwhelming, but the general takeaway is:

- In general, the more items we remove from the hash, the more properties are needed for the KEM scheme.
- The maximal items that can be removed from the hash "for free" are pw and (s, T). In other words, if τ' is defined as $H_{\mathsf{tag}}(pk, c, K)$ then no additional KEM properties are needed compared to $H_{\mathsf{tag}}(\mathsf{pw}, pk, s, T, c, K)$.
- Removing or not removing (s, T) does not make a difference, *except* that if pk is also removed, then we additionally need KBIND. (This is why we put all variants where (s, T) are removed in the appendices.)

Overall, we hope that this work will help the community converge upon a "right" version of OEKE. Since the password pw can be removed from the hash "for free", we advocate removing it for better defense against side-channel attacks. Whether other items can/should be removed depends on the underlying KEM scheme; in Section 2.3 we show that the Diffie–Hellman KEM scheme — with 1 excluded from valid ciphertexts — satisfies all properties we consider, but leave the analysis of post-quantum KEM schemes (or KEM schemes based on RSA) for future work.

1.3 How to Read This Paper

I certainly don't expect anyone to read through all 108 pages of this paper. (Well, perhaps the 2-page table of contents and the 3-page references shouldn't count, but even then it's still over 100 pages.) As I said right before Section 1.1, I have organized the paper in such a way that you should gain an intermediate

level of understanding of the technical details after reading Sections 1 and 2. In particular, Sections 2.1 and 2.2 explain why certain KEM properties are needed in certain OEKE variants.

Below I discuss who should read further:

- If you don't care about all those variants but would like to see a detailed security proof for the basic "hash everything" version, read Section 4.
- If, after reading Section 4, you'd like to additionally see the security proofs for the variants where pw, pk and/or c are removed from the hash, read Section 5. Note, however, that I assume familiarity with Section 4 there: I really cannot afford repeating everything in the proof every time I analyze a new variant. I stress again that reading Section 2 alone should be enough for being convinced that certain OEKE variants need certain KEM properties.
- If you'd like to additionally understand the variants where (s, T) are removed from the hash, my suggestion is to read Appendices B.1.1, B.2.1, B.3.1 and B.4.1. Basically removing (s, T) results in some non-attacks that the UC simulator has to take care of, which are explained in the aforementioned sections;[7] the actual security proofs in Appendices B.1.2, B.2.2, B.3.2 and B.4.2 are gross and provide little insight.
- If your first exposure to the security proof for OEKE was [ABJ25] (resp. [JRX25]), read Appendix A (resp. Appendix C): they provide high-level comparisons between the security proofs in this paper and in prior works.
- I don't know if anyone will be interested in the future directions mentioned in Section 6, but if so, then go for it.

2 Preliminaries and Technical Overview

Let n be the security parameter; we assume that all algorithms take 1^n as an input, and do not explicitly write it.

Key Encapsulation Mechanisms (KEMs). A *Key Encapsulation Mechanism (KEM)* scheme consists of three algorithms:

- The *key generation* algorithm Gen outputs a public/secret key pair (pk, sk). We write $(pk, sk) \leftarrow \mathsf{Gen}$, and use $\mathcal{PK}$ to denote the space of the public key pk.
- The *encapsulation* algorithm Encap takes as input a public key pk, and outputs a ciphertext c and a KEM key K. We write $(c, K) \leftarrow \mathsf{Encap}(pk)$.

[7] Except that if pk is also removed from the hash, you need the KBIND property for the KEM scheme, but that's explained in Section 2.2.

- The (deterministic) *decapsulation* algorithm Decap takes as input a secret key sk and a ciphertext c, and outputs a KEM key K. We write $K := \mathsf{Decap}(sk, c)$ or $\mathsf{Decap}(sk, c) = K$.

We require two syntactic properties for a KEM scheme:

- *Correctness:* We have

$$\Pr\left[\begin{array}{ccc} & & (pk, sk) \leftarrow \mathsf{Gen} \\ K = K' & : & (c, K) \leftarrow \mathsf{Encap}(pk) \\ & & K' := \mathsf{Decap}(sk, c) \end{array}\right] \geq 1 - \varepsilon_{\text{correctness}},$$

  where $\varepsilon_{\text{correctness}}$ is negligible.
  Note that the probability is taken over the randomness of both Gen and Encap, i.e., we allow for a negligible portion of "bad" (pk, sk) pairs under which correctness holds with low probability.

- *High min-entropy:* For any $pk' \in \mathcal{PK}$, we have

$$\Pr[pk = pk' : (pk, \star) \leftarrow \mathsf{Gen}] \leq p_{\text{guess}},$$

  where p_{guess} is negligible. In other words, a (computationally unbounded) adversary has negligible probability of guessing an honestly generated public key correctly.

Whenever we refer to a KEM scheme below, we always assume these two properties and do not explicitly mention them.

The most commonly seen security properties in the KEM literature are indistinguishability (IND-CPA, or just security) and indistinguishability under chosen-ciphertext attacks (IND-CCA, or just CCA), but the KEM scheme in OEKE needs other properties. The "hash everything" version of OEKE only requires some relatively standard properties for the underlying KEM scheme, which we define in Section 2.1. The variants where certain items are removed from the hash require a number of non-standard properties, which we define in Section 2.2.

In all KEM properties below, we use $\mathbf{Adv}_{\mathcal{A}}^{\text{property}}$ to denote the adversary $\mathcal{A}$'s advantage in breaking the property (either $\mathcal{A}$'s winning probability in the game, or $\mathcal{A}$'s distinguishing advantage between two games), and do not repeat it in each definition.

2.1 KEM Properties Used in the "Hash Everything" Version

Pseudorandom public keys. This property roughly says that an honestly generated public key is indistinguishable from uniformly random.

Definition 1. A KEM scheme has *pseudorandom public keys (UNI-PK)* if for any PPT adversary $\mathcal{A}$, $\Pr[b^* = 1]$ in the following two games are negligibly

close:[8]

challenge bit $b = 0$:	challenge bit $b = 1$:
$(pk, \star) \leftarrow \mathsf{Gen}$	$pk \leftarrow \mathcal{PK}$
$b^* \leftarrow \mathcal{A}(pk)$	$b^* \leftarrow \mathcal{A}(pk)$

This property is the same as what's called UNI-PK in [ABJ25, Definition 5]. It is needed because otherwise — as an extreme example, suppose Gen is so biased that half of the elements in $\mathcal{PK}$ (call the set of them U) are unreachable by Gen — the OEKE adversary[9] $\mathcal{A}$ can run an offline dictionary attack as follows: $\mathcal{A}$ tries to decrypt $\mathsf{2F}^{-1}(\mathsf{pw}^*, (s, T))$ using various pw^* values, and if the result is in U then $\mathcal{A}$ knows that pw^* is not the real password pw. In this way $\mathcal{A}$ can rule out half of all candidate passwords in expectation.

One-wayness and a variant. This property roughly says that given a public key and a ciphertext, an adversary cannot feasibly find the key encapsulated in the ciphertext.

Definition 2. A KEM scheme is *one-way (OW)* if for any PPT adversary $\mathcal{A}$, the probability that $\mathcal{A}$ wins the following game is negligible:

$(pk, sk) \leftarrow \mathsf{Gen}$
$(c, K) \leftarrow \mathsf{Encap}(pk)$
$K^* \leftarrow \mathcal{A}(pk, c)$
$\mathcal{A}$ wins if $K^* = K$

We also need a stronger variant in which the adversary is additionally given one-time access to a *plaintext-checking oracle*, which tells the adversary whether a ciphertext decapsulates to a certain key.

Definition 3. The definition of *one-wayness under one plaintext-checking attack (OW-1PCA)* is the same as that of OW, except that the adversary $\mathcal{A}$ is additionally given access to an oracle $\mathsf{PC}_{sk}(\star, \star)$, which on input $(c' \neq c, K')$ outputs 1 if $K' = \mathsf{Decap}(sk, c')$ and 0 otherwise. $\mathcal{A}$ is allowed to query this oracle only once.

This property is the same as what's called OW-PCA in [ABJ25, Definition 3], except that in our notion (1) the adversary can query its plaintext-checking oracle only once, and (2) the adversary is prohibited from querying its plaintext-checking oracle on the challenge ciphertext. The plaintext-checking oracle was first studied in [ABP15] in the context of public-key encryption schemes.

OW-1PCA is needed in the scenario where the OEKE adversary $\mathcal{A}$ forwards the P-to-P' message (s, T) without modification, but then modifies the P'-to-P message (c, τ') to some other (c^*, τ^*). Consider the time when P' outputs its

[8] The challenge bit b is not actually used in the definition, but it will come in handy in the security proofs later (we sometimes need concise names for these security games).

[9] All attacks in this section work even for OEKE with IC, so we use "OEKE adversary" rather than "OEKE-2F adversary".

session key SK': at this point $\mathcal{A}$ has not done any attack yet, so in the ideal world the PAKE functionality will set SK' at random. This means that $\mathcal{A}$ must not be able to compute K' (otherwise it can query $H_{\mathsf{key}}(K')$ and distinguish SK' from random); furthermore, forward secrecy requires that $\mathcal{A}$ should not compute K' even if it learns the password pw later (which would allow $\mathcal{A}$ to compute $pk := \mathsf{2F}^{-1}(\mathsf{pw}, (s, T))$) — which is why we give the public key to the adversary in the security game.

Furthermore, later when $\mathcal{A}$ modifies (c, τ') to $(c^* \neq c, \tau^* = H_{\mathsf{tag}}(\ldots, K^*))$ for K^* of its choice, it can check if $K^* = \mathsf{Decap}(sk, c^*)$ by observing if P's session key $SK = H_{\mathsf{key}}(K^*)$. This corresponds to the (single) query to the plaintext-checking oracle. (This requires that P and P''s passwords match; otherwise sk on the P side does not correspond to pk' on the P' side.)

Anonymity and a variant. This property roughly says that given a public key and a ciphertext, an adversary cannot feasibly tell whether the ciphertext corresponds to this public key or some other (freshly generated) public key.

Definition 4. A KEM scheme is *anonymous (ANO)* if for any PPT adversary $\mathcal{A}$, $\Pr[b^* = 1]$ in the following two games are negligibly close:

challenge bit $b = 0$:
$(pk, sk) \leftarrow \mathsf{Gen}$
$(c, \star) \leftarrow \mathsf{Encap}(pk)$
$b^* \leftarrow \mathcal{A}(pk, c)$

challenge bit $b = 1$:
$(pk, sk) \leftarrow \mathsf{Gen}$
$(pk', \star) \leftarrow \mathsf{Gen}$
$(c, \star) \leftarrow \mathsf{Encap}(pk')$
$b^* \leftarrow \mathcal{A}(pk, c)$

Similar to OW-1PCA, we also need the 1PCA variant of ANO.

Definition 5. The definition of *anonymity under one plaintext-checking attack (ANO-1PCA)* is the same as that of ANO, except that the adversary $\mathcal{A}$ is additionally given access to the oracle $\mathsf{PC}_{sk}(\star, \star)$, which on input $(c' \neq c, K')$ outputs 1 if $K' = \mathsf{Decap}(sk, c')$ and 0 otherwise. $\mathcal{A}$ is allowed to query this oracle only once.

This property is the same as what's called ANO-PCA in [ABJ25, Definition 4], except that in our notion (1) the adversary can query its plaintext-checking oracle only once, and (2) the adversary is not given pk' (whereas in their notion pk' is generated and given to the adversary in both games).

ANO-1PCA is needed again in the scenario where the OEKE adversary $\mathcal{A}$ forwards the P-to-P' message (s, T) without modification, but then modifies the P'-to-P message (c, τ'). Consider c: in the real world c is the result of $\mathsf{Encap}(pk)$, whereas in the ideal world the simulator does not know pk' at this point (it learns no information about pw, so it cannot decrypt (s, T) and learn pk) and has to simulate c as the result of $\mathsf{Encap}(pk')$ for a fresh pk'. $\mathcal{A}$ should not be able to tell these two cases apart (even after it learns pw later and uses it to compute $pk := \mathsf{2F}^{-1}(\mathsf{pw}, (s, T))$), which is ANO. 1PCA is needed for the same reason as in OW-1PCA.

Anonymity with uniform public keys and random oracles. Finally, we define a highly tailored KEM property that is implied by the preceding more standard properties. This property is for facilitating the security proof for OEKE-2F only; we do not expect it to be of independent interest.

Definition 6. A KEM scheme is *anonymous with q uniform public keys and random oracles (q-UNI-ANO-RO)* if $\Pr[b^* = 1]$ in the following two games are negligibly close:

challenge bit $b = 0$:	challenge bit $b = 1$:
$pk_1, \ldots, pk_q \leftarrow \mathcal{PK}$	$pk', pk_1, \ldots, pk_q \leftarrow \mathcal{PK}$
$i \leftarrow \mathcal{A}(pk_1, \ldots, pk_q)$	$i \leftarrow \mathcal{A}(pk_1, \ldots, pk_q)$
$(c, K) \leftarrow \mathsf{Encap}(pk_i)$	$(c, \star) \leftarrow \mathsf{Encap}(pk')$
$h_1 := H_1(K); h_2 := H_2(K)$	$h_1 \leftarrow \{0,1\}^n; h_2 \leftarrow \{0,1\}^{2n}$
$b^* \leftarrow \mathcal{A}(c, h_1, h_2)$	$b^* \leftarrow \mathcal{A}(c, h_1, h_2)$

Here, H_1 and H_2 are ROs with range $\{0,1\}^n$ and $\{0,1\}^{2n}$, respectively.

This property emerges when the OEKE adversary $\mathcal{A}$ sends (s^*, T^*) to P' which contains a wrong password guess. In the real world, (s^*, T^*) decrypt to a uniformly random KEM public key, which is then used to compute c, SK, τ (the $b = 0$ game). In simulation, we want to remove the decryption process and outright use a freshly sampled public key to compute c, and sample a uniformly random τ'; furthermore, the UC PAKE functionality will sample a uniformly random SK' (the $b = 1$ game). (h_1, h_2 in Definition 6 correspond to SK', τ' in the protocol, respectively.) What complicates things is that $\mathcal{A}$ can decrypt a large number of (s^*, T^*) pairs *under both correct and incorrect passwords*, resulting in a large number of pk^* values, *before* sending (s^*, T^*) to P' (or even before initiating P''s session); at this time the reduction does not know which (s^*, T^*) will be used by $\mathcal{A}$, or which password is correct, yet it has to program the public key (the "right" one that will be used in P''s algorithm) into the decryption process. As such, we need to give the reduction polynomially many public keys.

It is not hard to see that q-UNI-ANO-RO is implied by more standard properties:

Lemma 1. *Suppose the UNI-PK, OW and ANO properties hold for a KEM scheme. Then for any polynomial q, the q-UNI-ANO-RO property also holds for the KEM scheme.*

The proof is deferred to Appendix D.1.

2.2 Additional KEM Properties Used in Other Versions

Ciphertext binding. This property roughly says that given an honest tuple of (pk, c, K), an adversary cannot feasibly find another valid (pk, c^*, K) tuple (i.e., a new ciphertext that works for the same public key and the same KEM

key). In other words, the public key/KEM key pair (pk, K) binds a ciphertext. This essentially requires that the adversary cannot re-randomize a ciphertext it sees.

Definition 7. A KEM scheme is *ciphertext binding (CBIND)* if for any PPT adversary $\mathcal{A}$, the probability that $\mathcal{A}$ wins the following game is negligible:

$(pk, sk) \leftarrow \mathsf{Gen}$
$(c, K) \leftarrow \mathsf{Encap}(pk)$
$c^* \leftarrow \mathcal{A}(pk, c, K)$
abort if $c^* = c$
$\mathcal{A}$ wins if $\mathsf{Decap}(sk, c^*) = K$

CBIND is needed in OEKE variants where c is removed from H_{tag} inputs. Then we must prevent the OEKE adversary $\mathcal{A}$ from forwarding the P-to-P' message (s, T) without modification, but then modifying the P'-to-P message (c, τ') to some (c^*, τ') which makes P's check pass.[10] (If c is included in H_{tag} then P's check will not pass unless $(\dots, c, \dots)$ and $(\dots, c^*, \dots)$ form a collision for H_{tag}.) This should not happen even after $\mathcal{A}$ later learns pw and uses it to compute $pk := \mathsf{2F}^{-1}(\mathsf{pw}, (s, T))$.

In the actual definition of CBIND we additionally give K to the adversary. This is because the *reduction* in our OEKE security proof (see Claim 2) needs the KEM key to simulate P''s session key SK' and the authenticator τ'. It is possible that a reduction to a weaker CBIND property (where K is not given the adversary) can go through by sampling SK', τ' at random, but that would require changing the order of the games in the security proof, and we do not explore this possibility.

Remark 4. The weaker variant of CBIND, where K is not given to the adversary, is implied by OW-1CCA (which is arguably a more standard property): the reduction $\mathcal{R}_{\text{OW-1CCA}}$, on input (pk, c), invokes the CBIND adversary $\mathcal{A}$ on the same input; when $\mathcal{A}$ outputs $c^* \neq c$, $\mathcal{R}_{\text{OW-1CCA}}$ queries its decapsulation oracle on c^* and outputs the result. Hence, it is meaningful to study whether the security proof goes through under this weak CBIND property.

Unpredictability. This property roughly says that an adversary $\mathcal{A}$ cannot feasibly output a ciphertext/KEM key pair (c^*, K^*) "out of thin air", such that (pk, c^*, K^*) is a valid tuple for an honestly generated public key pk that $\mathcal{A}$ does not know.

[10] It is, in fact, unclear whether this is an "actual" attack: $\mathcal{A}$ causes the two parties to output the same session key by modifying c and forwarding all other protocol messages, but $\mathcal{A}$ does not learn the session key or the password. However, at least such an $\mathcal{A}$ makes the UC simulation fail, since the simulator is not able to tell that $\mathcal{A}$ is sending a valid c^* (it does not know sk). The same comment is true for the attacks in the "key binding" paragraph below.

Definition 8. A KEM scheme is *unpredictable (UNPRE)* if for any PPT adversary $\mathcal{A}$, the probability that $\mathcal{A}$ wins the following game is negligible:

$$\boxed{\begin{array}{l}(\star, sk) \leftarrow \mathsf{Gen} \\ (c^*, K^*) \leftarrow \mathcal{A}() \\ \mathcal{A} \text{ wins if } \mathsf{Decap}(sk, c^*) = K^*\end{array}}$$

A similar property for public-key encryption schemes, called decryption robustness, was first studied in [AMO+26].

UNPRE is needed in OEKE variants where pk is removed from H_{tag} inputs. If the OEKE adversary $\mathcal{A}$ can come up with c^*, K^* that break the UNPRE property, then it can compute $\tau^* := H_{\mathsf{tag}}(\dots, K^*)$ and send (c^*, τ^*) to P. This causes P's check to pass, and P will output $SK := H_{\mathsf{key}}(K^*)$, which $\mathcal{A}$ can compute. If pw is also removed from H_{tag} inputs then $\mathcal{A}$ can predict P's session key without knowing the password, which clearly breaks the security of OEKE. (This is the attack in Section 1.1.) If pw is included in H_{tag} then this attack requires $\mathcal{A}$ to know pw, so $\mathcal{A}$ is not "actually" breaking the security of OEKE, but at least the UC simulator cannot tell that $\mathcal{A}$ is performing this attack (it cannot decide whether $\mathsf{Decap}(sk, c^*) = K^*$), so it is unclear how the simulator should proceed in this case. (If pk is included in H_{tag} then the attack above does not work, since $\mathcal{A}$ cannot compute the correct $\tau^* = H_{\mathsf{tag}}(\dots, pk, \dots)$ without knowing pk.)

Remark 5. [ABJ25] seems to have noticed that some form of unpredictability is needed if pk is removed from H_{tag} inputs. [ABJ25, Section 5] says:

> *We note that, although it is tempting to remove pk from the H_{tag} input by arguing that $(\mathsf{pw}, [(s, T)])$ fix a single pk following the IC intuition. However [sic], this gives rise to cases in the proof where the adversary may try to break the protocol by delivering ciphertexts that are valid for all public keys. It is likely that these cases can be excluded by assuming additional properties on the KEM, but we choose not to do so.*

It appears to us that "delivering ciphertexts that are valid for all public keys" should be "finding a ciphertext/KEM key pair that is valid for all public keys, and delivering the ciphertext". That is, if one can find a "special" pair of c^* and K^* such that $\mathsf{Decap}(sk, c^*) = K^*$ for every valid secret key sk, then the UNPRE property trivially breaks down, and the OEKE adversary can run the aforementioned attack.

Key binding. This property roughly says that given an honest tuple of (pk', c, K') and another honestly generated public key pk, an adversary cannot feasibly come up with a ciphertext c^* such that (pk, c^*, K') is also a valid tuple. In other words, the KEM key K' binds a public key/ciphertext pair, although we give the adversary the power of choosing one ciphertext only, and both public keys and the other ciphertext are simply given to the adversary. (In the actual

definition, we let pk' be sampled uniformly at random, rather than honestly generated.)

Definition 9. A KEM scheme is *key binding (KBIND)* if for any PPT adversary $\mathcal{A}$, the probability that $\mathcal{A}$ wins the following game is negligible:

$(pk, sk) \leftarrow \mathsf{Gen}$
$pk' \leftarrow \mathcal{PK}$
$(c, K') \leftarrow \mathsf{Encap}(pk')$
$c^* \leftarrow \mathcal{A}(pk, pk', c, K')$
$\mathcal{A}$ wins if $\mathsf{Decap}(sk, c^*) = K'$

KBIND is needed in OEKE variants where pw, pk and c are *all* removed from H_{tag} inputs. The attack only works in the setting where P's password pw and P''s password pw' are different (in prior arguments under KEM properties we implicitly assumed that $\mathsf{pw} = \mathsf{pw}'$). It is more subtle than previous attacks, so let us describe the OEKE adversary $\mathcal{A}$ in more detail:

1. On protocol message (s, T) from P, forward (s, T) to P' without modification. This causes P' to decrypt $pk' := \mathsf{2F}^{-1}(\mathsf{pw}', (s, T))$, which is uniformly random and independent of pk since $\mathsf{pw}' \neq \mathsf{pw}$. P' then sends $(c, \tau' := H_{\mathsf{tag}}(s, T, K'))$ to P (intercepted by $\mathcal{A}$) and outputs $SK' := H_{\mathsf{tag}}(K')$.

2. Suppose $\mathcal{A}$ now learns pw' (i.e., after P''s session completes); then $\mathcal{A}$ can use it to recover pk'. If $\mathcal{A}$ can find another c^* that breaks the KBIND property, then $\mathcal{A}$ can send (c^*, τ') to P. P will compute the same K' upon decapsulation, so its check $\tau' \stackrel{?}{=} H_{\mathsf{tag}}(s, T, K')$ will pass and P will output the same $H_{\mathsf{tag}}(K')$ as P' does.

Similar to the attack in the "ciphertext binding" paragraph, $\mathcal{A}$ causes the two parties to output the same session key by modifying c and forwarding all other protocol messages. (If pw is included in H_{tag} then P's check will not pass unless $(\mathsf{pw}, \dots)$ and $(\mathsf{pw}', \dots)$ form a collision for H_{tag}. If pk is included in H_{tag} then P's check will not pass unless either (1) the uniformly random and independent pk' happens to be equal to pk, or (2) $(\dots, pk, \dots)$ and $(\dots, pk', \dots)$ form a collision for H_{tag}. Similarly, if c is included in H_{tag} then P's check will not pass unless $(\dots, c, \dots)$ and $(\dots, c^*, \dots)$ form a collision for H_{tag}.)

Furthermore, KBIND is also needed if pk and (s, T) are *both* removed from H_{tag} inputs. The attack goes as follows (here pw and pw' need to be the same if pw is included in H_{tag}):

1. Ignore protocol message (s, T) from P, and send random junk $(s^*, T^*) \neq (s, T)$ to P'. This causes P' to decrypt $pk' := \mathsf{2F}^{-1}(\mathsf{pw}, (s^*, T^*))$, which is uniformly random and independent of pk. P' then sends $(c, \tau' := H_{\mathsf{tag}}(\mathsf{pw}, K'))$ to P (intercepted by $\mathcal{A}$) and outputs $SK' := H_{\mathsf{tag}}(K')$.

2. Suppose $\mathcal{A}$ now learns pw' (i.e., after P''s session completes); then $\mathcal{A}$ can use it to recover pk'. If $\mathcal{A}$ can find another c^* that breaks the KBIND

property, then $\mathcal{A}$ can send (c^*, τ') to P. P will compute the same K' upon decapsulation, so its check $\tau' \stackrel{?}{=} H_{\mathsf{tag}}(\mathsf{pw}, K')$ will pass and P will output the same $H_{\mathsf{tag}}(K')$ as P' does.

In this attack, $\mathcal{A}$ modifies the P-to-P' message (s, T), but still forwards τ'. Again, this poses a problem to the UC simulator since the simulator cannot tell if $\mathcal{A}$ is sending a valid c^* or not (it does not know sk). (If pk is included in H_{tag} then P's check will not pass unless either (1) the uniformly random and independent pk' happens to be equal to pk, or (2) $(\dots, pk, \dots)$ and $(\dots, pk', \dots)$ form a collision for H_{tag}. If (s, T) are included in H_{tag} then P's check will not pass unless $(\dots, s, T, \dots)$ and $(\dots, s^*, T^*, \dots)$ form a collision for H_{tag}.)

Note that the OEKE adversaries above only know pk' and c, whereas in the definition of KBIND we additionally give pk and K' to the adversary. This is because the *reduction* in our OEKE security proof (see Claim 4) needs pk to simulate the P-to-P' message (s, T), and needs K' to simulate P''s session key SK' and the authenticator τ' (cf. the "ciphertext binding" paragraph).

We also consider a much weaker variant of KBIND, where the adversary $\mathcal{A}$ is forced to output $c^* = c$. Since $\mathcal{A}$ is passive now, this property can be described without mentioning an adversary, and we show that it is implied by UNI-PK and OW:

Lemma 2. *Suppose the UNI-PK and OW properties hold for a KEM scheme. Then*

$$\Pr\left[\; K = K' \;:\; \begin{array}{c} (\star, sk) \leftarrow \mathsf{Gen} \\ pk' \leftarrow \mathcal{PK} \\ (c, K') \leftarrow \mathsf{Encap}(pk') \\ K := \mathsf{Decap}(sk, c) \end{array} \right]$$

is negligible.

The proof is straightforward and is deferred to Appendix D.2.

Lemma 2 suggests that assuming UNI-PK and OW, we can strengthen KBIND so that $\mathcal{A}$ is required to output $c^* \neq c$.

Remark 6. Consider a(nother) weaker variant of KBIND, where K' is not given to the adversary; and a stronger variant of ANO-1PCA, where pk' is additionally given to the adversary. Then this weak KBIND is implied by UNI-PK + strong ANO-1PCA + CBIND via the following sequence of games:

0. The CBIND game.
1. Independently generate pk' and additionally give pk' to $\mathcal{A}$ (pk' is not used anywhere else).
2. Instead of computing $(c, K) \leftarrow \mathsf{Encap}(pk)$, compute $(c, K') \leftarrow \mathsf{Encap}(pk')$ and give (pk, pk', c) to $\mathcal{A}$. $\mathcal{A}$'s winning probability is negligibly close due to the ANO-1PCA property.

3. Instead of generating pk' honestly, sample $pk' \leftarrow \mathcal{PK}$. $\mathcal{A}$'s winning probability is negligibly close due to the UNI-PK property. At this point we arrive at the KBIND game.

A formal proof of this is left to the reader as an exercise (think about why 1PCA is needed in step 2).

Collision resistance. This property roughly says that given two honestly generated public keys pk, pk', an adversary cannot feasibly come up with a ciphertext/KEM key pair c^*, K^* such that both (pk, c^*, K^*) and (pk', c^*, K^*) are valid tuples. In other words, an (adversarially generated) ciphertext/KEM key pair (c^*, K^*) binds an (honestly generated) public/secret key pair.

Definition 10. A KEM scheme is *collision resistant (CR)* if for any PPT adversary $\mathcal{A}$, the probability that $\mathcal{A}$ wins the following game is negligible:

$(pk, sk) \leftarrow \mathsf{Gen}$
$(pk', sk') \leftarrow \mathsf{Gen}$
$(c^*, K^*) \leftarrow \mathcal{A}(pk, pk')$
$\mathcal{A}$ wins if $\mathsf{Decap}(sk, c^*) = \mathsf{Decap}(sk', c^*) = K^*$

A very similar notion, called strong collision freeness (SCFR-CPA), has been studied in the context of public-key encryption schemes [GMP22]. However, the adversary in SCFR-CPA only needs to output a KEM ciphertext, whereas we additionally require the adversary to output the corresponding KEM key.

CR is needed in OEKE variants where pw and pk are *both* removed from H_{tag} inputs. This is to prevent the following attack: The OEKE adversary $\mathcal{A}$ completely disregards P', and on (s, T) from P, $\mathcal{A}$ chooses two password guesses $\mathsf{pw}_1^*, \mathsf{pw}_2^*$ and computes $pk_1^* := \mathsf{2F}^{-1}(\mathsf{pw}_1^*, (s, T)), pk_2^* := \mathsf{2F}^{-1}(\mathsf{pw}_2^*, (s, T))$. If $\mathcal{A}$ can find (c^*, K^*) that break the CR property for pk_1^*, pk_2^*, it then sends $(c^*, \tau^* = H_{\mathsf{tag}}(s, T, c^*, K^*))$ to P. As long as one of $\mathsf{pw}_1^*, \mathsf{pw}_2^*$ is equal to P's password pw, P's check will pass and $\mathcal{A}$ can compute P's session key $SK = H_{\mathsf{key}}(K^*)$. In this way $\mathcal{A}$ can make two password guesses per session, which is not allowed by PAKE security. (If either pw or pk is included in H_{tag} then the attack above does not work, since $\mathcal{A}$ needs to commit to a single (pw^*, pk^*) pair while sending τ^*, unless $\mathcal{A}$ finds a collision in H_{tag}.)

It is not hard to see that CR implies UNPRE:

Lemma 3. *Suppose the CR property holds for a KEM scheme. Then the UNPRE property also holds for the KEM scheme.*

The proof is deferred to Appendix D.3.

It is unclear whether our new KEM properties are of independent interest. Some of them, such as CR, appear natural enough; while properties such as KBIND look highly tailored (especially since in KBIND one of the public keys is generated honestly, and the other is sampled uniformly at random). Either way, we are not aware of any other applications of these properties yet.

2.3 Example: Diffie–Hellman KEM

It is instructive to see what cryptographic assumptions the various KEM properties are, in the well-known Diffie–Hellman KEM. This KEM scheme works in a finite cyclic group with (public) generator g and prime order q superpolynomial in n, and

- Gen samples $x \leftarrow \mathbb{Z}_q$, computes $X := g^x$, and outputs (X, x).
- $\mathsf{Encap}(X)$ samples $y \leftarrow \mathbb{Z}_q$, computes $Y := g^y$ and $K := X^y$, and outputs (Y, K).
- $\mathsf{Decap}(x, Y)$ outputs Y^x.

(Note that the computation of the ciphertext Y does not rely on the public key X; in terms of key exchange, Diffie–Hellman is 1-simultaneous round, whereas KEM can model any 2-message-flow key exchange protocol.) The Diffie–Hellman KEM is perfectly correct (i.e., $\varepsilon_{\text{correctness}} = 0$), and $p_{\text{guess}} = 1/q$. Furthermore, it satisfies various properties we have defined (below whenever we mention x, y, we assume they are sampled as $x, y \leftarrow \mathbb{Z}_q$):

- UNI-PK says that $X = g^x$ is indistinguishable from a uniformly random element of $\mathbb{G}$, which holds perfectly.
- OW says that given $(X = g^x, Y = g^y)$ it is infeasible to compute $K = g^{xy}$, which is the Computational Diffie–Hellman (CDH) assumption.
- OW-1PCA says that given $(X = g^x, Y = g^y)$ and one-time access to an oracle that on input $(Y' \neq Y, K')$ checks if $(Y')^x = K'$, it is infeasible to compute $K = g^{xy}$. This is the Gap Diffie–Hellman (GDH) assumption, except that (1) when the adversary makes a DH oracle query, its first input is fixed to be X, (2) when the adversary makes a DH oracle query, its second input must not be equal to Y, and (3) the adversary can query the (restricted) DH oracle only once.[11] This limited version of GDH can be reduced to CDH (on the single oracle query, the reduction to CDH returns 1 with probability 1/2 and 0 with probability 1/2; in this way the reduction has a 1/2 chance of simulating the game correctly).
- ANO says that $(X = g^x, Y = g^y)$ and $(X' = g^{x'}, Y = g^y)$ for another $x' \leftarrow \mathbb{Z}_q$ are indistinguishable, which holds perfectly. (This property holds perfectly for any KEM scheme where the ciphertext does not rely on the public key.) It of course holds perfectly even if the adversary is additionally given the plaintext-checking oracle.

[11] GDH with restriction (1) in place, namely the oracle only takes two inputs (Y', K') and checks if $K' = (Y')^x$, is sometimes called "Strong Diffie–Hellman (SDH)" [ABR01, Definition 9]. (Some works, e.g., [LLMT25, Section 2], use the term "GDH" to refer to SDH.) Restriction (2) is redundant since a query on (Y, K') is equivalent to a query on $(Y \cdot g, K' \cdot X)$. As such, this assumption could be properly named "1SDH".

- CBIND says that given $(X = g^x, Y = g^y)$ it is infeasible to come up with $Y^* \neq Y$ such that $(Y^*)^x = X^y$, which holds statistically (such Y^* does not exist unless $x = 0$, which happens with negligible probability $1/q$).

- UNPRE says that it is infeasible to come up with (Y^*, K^*) such that $K^* = (Y^*)^x$. **This property does NOT hold for the Diffie–Hellman KEM scheme as described above**, since the adversary can simply output $(Y^* = 1, K^* = 1)$. (This is what caused us trouble in Section 1.1.) To fix this, we have to dictate that the identity group element *not* be a valid ciphertext, i.e., $\mathsf{Decap}(\star, 1)$ must output $\perp$. Then UNPRE is a statistical property.

- KBIND says that given $(X = g^x, X', Y = g^y, (X')^y)$ where X' is a freshly sampled random group element, it is infeasible to come up with Y^* such that $(Y^*)^x = (X')^y$. We claim that this is implied by the CDH assumption. Since $x = 0$ with negligible probability $1/q$, below we assume this does not happen, and the adversary's task is to compute $((X')^y)^{\frac{1}{x}}$. Let $Z = (X')^y$, and consider a stronger adversary that is additionally given $\log_g Z$. Then computing $Z^{\frac{1}{x}} = (g^{\frac{1}{x}})^{\log_g Z}$ is equivalent to computing $g^{\frac{1}{x}}$. But computing $g^{\frac{1}{x}}$ given $X = g^x$ ($\mathcal{A}$'s other inputs are independent of X and are thus useless) is the inverse Diffie–Hellman problem, which is well-known to be equivalent to CDH.[12]

- CR says that given $(X = g^x, X' = g^{x'})$ for $x' \leftarrow \mathbb{Z}_q$ independent of x, it is infeasible to come up with (Y^*, K^*) such that $(Y^*)^x = (Y^*)^{x'} = K^*$. Similar to UNPRE, this property does NOT hold for the Diffie–Hellman KEM scheme as described above, since the adversary can simply output $(Y^* = 1, K^* = 1)$. Again, we need to exclude 1 from valid ciphertexts. Then CR is a statistical property: such Y^* does not exist unless $x = x'$, which happens with negligible probability $1/q$.

Overall, we conclude that the Diffie–Hellman KEM scheme has all desired properties under the CDH assumption, *except for UNPRE and CR where we need to exclude* 1 *from valid ciphertexts.*

Remark 7. The arguments in the CBIND, UNPRE, KBIND and CR paragraphs require the group order to be prime. However, they can be straightforwardly extended to the general case where the group order has at least one large prime factor (a requirement we have to make anyway, since otherwise discrete logarithm is easy in the group due to the Pohlig–Hellman algorithm) — *except* that for UNPRE and CR, we need to additionally exclude all elements of the small subgroup(s) from valid ciphertexts (e.g., if $|\mathbb{G}| = 2p$ for prime p, then g^p cannot be a valid ciphertext, since otherwise the adversary can output $(Y^* = g^p, K^* = 1)$ and win the UNPRE game with probability 1/2).

[12] Inverse Diffie–Hellman is equivalent to square Diffie–Hellman [BDZ03, Section 2.2], which in turn is equivalent to CDH [BDZ03, Section 2.1].

2.4 The UC PAKE Functionality

We define the UC PAKE with initiator-side explicit authentication functionality $\mathcal{F}_{\text{PAKE-1EA}}$ in Figure 4. Most of it is identical to the standard UC PAKE functionality (with implicit authentication) in [CHK+05]; the differences are highlighted in blue.

- On input $(\mathsf{NewSession}, sid, P, P', \mathsf{pw}, \text{role})$ from P, send $(\mathsf{NewSession}, sid, P, P')$ to $\mathcal{S}$. Furthermore, if this is the first NewSession message for sid, or this is the second NewSession message for sid and there is a record $\langle P', P, \star, \star\rangle$, then store $\langle P, P', \mathsf{pw}, \text{role}\rangle$ and mark it fresh. // P's counterparty is P', its password is pw, and its role is "role"
- On $(\mathsf{TestPwd}, sid, P, \mathsf{pw}^*)$ from $\mathcal{S}$, if there is a record $\langle P, P', \mathsf{pw}, \star\rangle$ marked fresh, then do:
  1. If $\mathsf{pw}^* = \mathsf{pw}$, then mark the record compromised and send "correct guess" to $\mathcal{S}$.
  2. If $\mathsf{pw}^* \neq \mathsf{pw}$, then mark the record interrupted and send "wrong guess" to $\mathcal{S}$.
- On $(\mathsf{NewKey}, sid, P, SK^* \in \{0,1\}^n)$ from $\mathcal{S}$, if there is a record $\langle P, P', \mathsf{pw}, \text{role}\rangle$, and this is the first NewKey message for sid and P, then output (sid, SK) to P, where SK is defined as follows:
  1. If the record is compromised, then set $SK := SK^*$. // Successful online attack. This is the only case where SK^* matters; in the two cases below the functionality ignores SK^* and defines SK independently
  2. If the record is fresh, a session key (sid, SK') has been output to P', at which time there was a record $\langle P', P, \mathsf{pw}, \star\rangle$ marked fresh, then set $SK := SK'$. // If no attack in this session in either direction, and passwords match, then session keys should also match
  3. Otherwise, if role = "respondent", sample $SK \leftarrow \{0,1\}^n$; if role = "initiator", set $SK := \perp$.

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

Figure 4: UC PAKE with initiator-side explicit authentication functionality $\mathcal{F}_{\text{PAKE-1EA}}$. Differences with the standard UC PAKE functionality in [CHK+05] are highlighted in blue

We refer the reader to [Xu25, Section 2.1] for an explanation of the standard UC PAKE functionality; here we only explain the differences (the blue text). The two protocol parties are called "initiator" and "respondent": the initiator sends the first protocol message, which the respondent waits for before it

sends its own message. In order to match the OEKE protocol, our functionality achieves explicit authentication for the initiator side only; that is, if the initiator's algorithm does not result in a correct execution, it will output $\perp$ instead of an n-bit session key. There are three scenarios of an incorrect execution:

1. The MitM adversary performs a failed online attack on the initiator (i.e., the initiator's session record is interrupted).
2. The adversary does not perform an online attack on the initiator (i.e., the initiator's session record is fresh), but performs a (successful or failed) online attack on its counterparty.
3. The adversary does not perform an online attack on either party, but the two parties' passwords are different.

All these three possibilities are covered by the third bullet under NewKey,[13] in which case we let the initiator output $\perp$. There is no change from the standard UC PAKE functionality on the respondent side, since the respondent always outputs an n-bit session key (i.e., the respondent only performs implicit authentication).

Remark 8. [BCP+23, Fig.4] also gives a UC PAKE-1EA functionality; however, their functionality is problematic and cannot be realized by (any version of) OEKE. The issue is that their functionality treats the three incorrect execution scenarios above differently: in case (1) it lets the initiator output error, in case (3) it lets the initiator output a uniformly random session key, and the paper does not seem to describe what the functionality should do in case (2). In the real-world protocol, the initiator can tell that authentication fails but cannot tell which is the case, so it has to either output a uniformly random session key *in all cases* (which is the standard PAKE functionality) or output $\perp$ *in all cases* (which is our PAKE-1EA functionality).

3 UC Description of the OEKE-2F Protocol

We give a formal, UC-compatible description of the OEKE-2F protocol in Figure 5. Note that UC PAKE considers the case that the two parties' passwords are different; therefore, we need to use different notations pw and pw' for the password of P and P', respectively.[14] We only describe the "hash everything" version of the protocol; in other variants one or more of pw, pk, (s, T) and c can be removed from the inputs to H_{tag}. We omit sid from RO inputs (e.g., we write "$H_1(\mathsf{pw}, r)$" rather than "$H_1(sid, \mathsf{pw}, r)$"), although we stress that sid

[13] The third bullet also covers the possibility that the adversary does not perform an online attack on party P (i.e., P's session record is fresh), and P outputs its session key before its counterparty does. However, in this case P cannot be the initiator, since the initiator outputs after the respondent.

[14] [ABJ25, Fig.4] uses pw for both parties' passwords, which is not the "right" protocol description in the UC setting.

must be included in order to achieve domain separation. We assume that *sid* contains the names of the two protocol parties.

Let $\mathcal{G}$ be a PPT algorithm that outputs a (multiplicative) Abelian group $\mathbb{G}$ whose order is superpolynomial, and where uniformly sampling, multiplication and inverse can be done in PPT. (In the case of Diffie–Hellman KEM we need $\mathbb{G}$ to be cyclic and $|\mathbb{G}|$ to be public, but we do not make such requirements in general.) We implicitly assume that $\mathbb{G} \leftarrow \mathcal{G}$ is run as part of the initialization process. Furthermore, we assume that all algorithms take $\mathbb{G}$ as an input, and do not explicitly write it.
Let $(\mathsf{Gen}, \mathsf{Encap}, \mathsf{Decap})$ be a KEM scheme whose public-key space $\mathcal{PK} = \mathbb{G}$.

Party P, step 1

On input $(\mathsf{NewSession}, sid, P', \mathsf{pw}, \text{“initiator”})$, generate $(pk, sk) \leftarrow \mathsf{Gen}$, sample $r \leftarrow \{0,1\}^{3n}$, compute

$$R := H_1(\mathsf{pw}, r), \qquad T := pk \cdot R,$$
$$t := H_2(\mathsf{pw}, T), \qquad s := r \oplus t,$$

and send (sid, s, T) to P'. Store $\langle \mathsf{pw}, pk, sk, s, T \rangle$ associated with *sid*.

Party P'

On input $(\mathsf{NewSession}, sid, P, \mathsf{pw}', \text{“respondent”})$ and (sid, s, T) from P, compute

$$t' := H_2(\mathsf{pw}', T), \qquad r' := s \oplus t',$$
$$R' := H_1(\mathsf{pw}', r'), \qquad pk' := T/R',$$

$(c, K') \leftarrow \mathsf{Encap}(pk')$, $SK' := H_{\mathsf{key}}(K')$, and $\tau' := H_{\mathsf{tag}}(\mathsf{pw}', pk', s, T, c, K')$, send (sid, c, τ') to P, and output (sid, SK').

Party P, step 2

On (sid, c, τ') from P', retrieve $\langle \mathsf{pw}, pk, sk, s, T \rangle$ associated with *sid*. Then compute $K := \mathsf{Decap}(sk, c)$, $\tau := H_{\mathsf{tag}}(\mathsf{pw}, pk, s, T, c, K)$, and

$$SK := \begin{cases} H_{\mathsf{key}}(K) & \text{if } \tau = \tau' \\ \bot & \text{if } \tau \neq \tau' \end{cases}$$

and output (sid, SK). // No need to store pw, pk or (s, T) in step 1 if not included in H_{tag} (but always need to store sk)

Figure 5: UC description of the OEKE-2F protocol (the “hash everything” version)

Correctness. Suppose $\mathsf{pw} = \mathsf{pw}'$ and there is no adversary. Then P' computes

$$t' = H_2(\mathsf{pw}', T) = t, \qquad r' = s \oplus t' = r,$$
$$R' = H_1(\mathsf{pw}', r') = R, \qquad pk' = T/R' = pk,$$

so (c, K') is the result of $\mathsf{Encap}(pk)$ for an honestly generated pk. Then on the P side, P computes $K := \mathsf{Decap}(sk, c)$, which is equal to K' except with probability $\varepsilon_{\text{correctness}}$ (the correctness error of the KEM scheme). If $K = K'$ then $\tau = \tau'$ so P's check passes, and P will output $SK = SK'$. Therefore, the OEKE protocol has correctness error $\varepsilon_{\text{correctness}}$.

4 Security Analysis of the "Hash Everything" Version

The main theorem in this section is:

Theorem 1. *Suppose the UNI-PK, OW-1PCA and ANO-1PCA properties hold for the KEM scheme. Then the OEKE-2F protocol as described in Figure 5 UC-realizes $\mathcal{F}_{\text{PAKE-1EA}}$ (Figure 4), in the random oracle model for H_1, H_2, H_{key} and H_{tag}.*

The rest of this section is dedicated to the proof of Theorem 1. In Section 4.1 we describe the simulator, and in Section 4.2 we argue why the simulator generates a view indistinguishable from the real-world view.

As standard in UC proofs, w.l.o.g. we only construct a simulator $\mathcal{S}$ for the "dummy" adversary $\mathcal{A}$ that merely forwards all messages between the environment $\mathcal{Z}$ and protocol parties [Can01, Claim 11]. We omit *sid* in protocol messages and honest parties' outputs (e.g., we write "P outputs SK" rather than "P outputs (sid, SK)").

4.1 The Simulator

4.1.1 Simulation Strategy

A large portion of the simulator is routine; below we explain the idea behind the non-trivial parts.

Simulating honest P-to-P' message. When P's session begins, $\mathcal{S}$ must simulate its message (s, T) to P'. This can be easily done: just sample (s, T) at random. However, every time $\mathcal{A}$ queries $H_2(\mathsf{pw}^*, T)$ (let the result be t^*), $\mathcal{S}$ needs to generate a KEM key pair (pk^*, sk^*) for later use (see the "extracting password guess from malicious P'" paragraph below) and make sure that (pk^*, sk^*) is consistent with the RO queries. This can be done as follows: $\mathcal{S}$ computes

$$r^* := s \oplus t^*, \quad R^* := T/pk^*,$$

and programs $H_1(\mathsf{pw}^*, r^*) := R^*$. This also allows $\mathcal{S}$ to store the association between pw^* and (pk^*, sk^*). (Note that (pk^*, sk^*) need to be freshly generated every time $\mathcal{A}$ queries $H_2(\star, T)$; otherwise R^* never changes and $\mathcal{A}$ can easily find collisions in H_1.)

Extracting password guess from malicious P. When the adversary $\mathcal{A}$ sends a message (s^*, T^*) to P', if it is different from the honest message (s, T), the simulator $\mathcal{S}$ must extract a password guess (if any) and send a corresponding $\mathsf{TestPwd}$ command to $\mathcal{F}_{\text{PAKE-1EA}}$. There are two types of attack, which need to be addressed separately:

1. *"Normal" online attack:* $\mathcal{A}$ executes P's algorithm on a password guess pw^*. That is, $\mathcal{A}$ generates KEM public key pk^*, samples randomness r^*, computes

$$R^* := H_1(\mathsf{pw}^*, r^*), \qquad T^* := pk^* \cdot R^*,$$
$$t^* := H_2(\mathsf{pw}^*, T^*), \qquad s^* := r^* \oplus t^*,$$

   and sends (s^*, T^*) to P'.
   It takes a while to realize how to extract pw^* from (s^*, T^*). First, $\mathcal{S}$ should scan all $H_2(\star, T^*)$ queries, which yields multiple pw^* values; $\mathcal{S}$ must decide which one is the "actual" password guess. This can be done via the following. For every $H_2(\mathsf{pw}^*, T^*)$ query (let the result be t^*), $\mathcal{S}$ computes the corresponding $r^* := s^* \oplus t^*$, and checks if (and when) $H_1(\mathsf{pw}^*, r^*)$ was queried. *With overwhelming probability, there is at most one such H_1 query made before the $H_2(\mathsf{pw}^*, T^*)$ query*, from which $\mathcal{S}$ can extract the "actual" password guess pw^*.

2. *Related key attack:* This attack works only after $\mathcal{A}$ receives an honest (s, T) pair from P. $\mathcal{A}$ skips generating pk^* or sampling r^*; instead, $\mathcal{A}$ picks any $\Delta \in \mathbb{G}$, computes $T^* := T \cdot \Delta$ and

$$t^* := H_2(\mathsf{pw}^*, T), \qquad (t^*)' := H_2(\mathsf{pw}^*, T^*),$$
$$\delta := t^* \oplus (t^*)', \qquad s^* := s \oplus \delta,$$

   and sends (s^*, T^*) to P'.
   Let us first explain what $\mathcal{A}$ is trying to achieve here. The same randomness r yields two valid messages, (s, T) and (s^*, T^*), such that

$$s \oplus s^* = H_2(\mathsf{pw}, T) \oplus H_2(\mathsf{pw}, T^*),$$

   which allows $\mathcal{A}$ to pick T^* and "reverse engineer" the corresponding s^* if its password guess $\mathsf{pw}^* = \mathsf{pw}$. (s^*, T^*) implicitly encrypts $pk^* = T^*/H_1(\mathsf{pw}, r)$, which $\mathcal{A}$ can compute as $pk \cdot (T^*/T)$.
   To extract pw^* from (s^*, T^*), $\mathcal{S}$ should scan all $H_2(\star, T)$ and $H_2(\star, T^*)$ queries (which yield multiple candidate pw^* values) and check which ones satisfy

$$H_2(\mathsf{pw}^*, T) \oplus H_2(\mathsf{pw}^*, T^*) = s \oplus s^*.$$

   With overwhelming probability, at most one pw^* will satisfy this equation.

Remark: On the term "anchor query". The term "anchor query" was coined in [MRR20, Section 3.3], where it refers to the special $H_1(\mathsf{pw}^*, r^*)$ query made before the $H_2(\mathsf{pw}^*, T^*)$ query in case (1) above; case (2) is completely missing in [MRR20]. This error is fixed in [JRX25], which uses "anchor query" to refer to something else: the $H_2(\mathsf{pw}^*, T^*)$ query in either case (1) or (2) (see [JRX25, Section 4.3.1] and [JRX25, Figure 11, step 4]). This term is not used in [ABJ25].

Extracting password guess from malicious P'. When the adversary $\mathcal{A}$ sends a message (c^*, τ^*) to P, if $\mathcal{A}$ is not passive (i.e., it is not the case that $(s^*, T^*) = (s, T)$ *and* $(c^*, \tau^*) = (c, \tau')$), the simulator $\mathcal{S}$ must extract a password guess (if any) and send a corresponding $\mathsf{TestPwd}$ command to $\mathcal{F}_{\text{PAKE-1EA}}$.

$\mathcal{S}$ should find the H_{tag} query resulting in τ^*, which contains $(\mathsf{pw}^*, pk^*, K^*)$. However, pw^* constitutes a password guess only if it is consistent with pk^* and K^*, i.e., $K^* = \mathsf{Decap}(sk^*, c^*)$ for sk^* corresponding to pk^*. $\mathcal{S}$ can check this condition because it has stored the correspondence between pw^* and (pk^*, sk^*) (see the "simulating honest P-to-P' message" paragraph above), and can use sk^* to check if the equation holds. (This extraction strategy does not work anymore if *either* pw^* *or* pk^* is not included in H_{tag}. In later sections we will explain how to simulate other versions of the protocol.)

4.1.2 Formal Description of the Simulator

Below we use $\mathcal{F}$ as a shorthand for $\mathcal{F}_{\text{PAKE-1EA}}$. Please see Figures 6 and 7 for the simulator $\mathcal{S}$.

As notational conventions, we write "$\mathcal{S}$ sends message m from P to P'" as a shorthand for "$\mathcal{S}$ sends m to $\mathcal{Z}$ that pretends to be a real-world protocol message from P to P', intercepted by $\mathcal{A}$ which in turn forwards it to $\mathcal{Z}$". Similarly, we write "$\mathcal{S}$ receives message m from $\mathcal{A}$ to P" for "$\mathcal{Z}$ instructs $\mathcal{A}$ to send m to P, and in the ideal world $\mathcal{A}$ does not exist and it is $\mathcal{S}$ who receives m from $\mathcal{Z}$".

Our simulator is, by and large, the same as the one in [ABJ25, Fig.5 and 6], although we completely rewrite it to improve readability. For those who would like to go over both our simulator and the one in [ABJ25], we include a comparison in Appendix A.

P-to-P' message

On $(\mathsf{NewSession}, sid, P, P',$ "initiator") from $\mathcal{F}$, sample $s \leftarrow \{0,1\}^{3n}, T \leftarrow \mathbb{G}$ and send (s, T) from P to P'.

P''s session key and P'-to-P message

On $(\mathsf{NewSession}, sid, P', P,$ "respondent") from $\mathcal{F}$ and (s^*, T^*) from $\mathcal{A}$ to P', simulate P''s output and the P'-to-P message as follows:

1. If $(s^*, T^*) = (s, T)$, send $(\mathsf{NewKey}, sid, P', 0^n)$ to $\mathcal{F}$; generate $(pk', \star) \leftarrow \mathsf{Gen}$, compute $(c, \star) \leftarrow \mathsf{Encap}(pk')$, sample $\tau' \leftarrow \{0,1\}^{2n}$, and send (c, τ') from P' to P.
2. Otherwise find $\mathcal{A}$'s $H_2((\mathsf{pw}')^*, T^*)$ query (let the result be t^*) that satisfies *either* of the following:
   - *Type (1):* $\mathcal{A}$ queried $H_1((\mathsf{pw}')^*, r^*)$ *before* the $H_2((\mathsf{pw}')^*, T^*)$ query, where $r^* = s^* \oplus t^*$.
   - *Type (2):* $\mathcal{A}$ also queried $H_2((\mathsf{pw}')^*, T)$, and
   $$s \oplus s^* = H_2((\mathsf{pw}')^*, T) \oplus t^*.$$

   (a) If there is more than one such $(\mathsf{pw}')^*$, abort.

   (b) If there is exactly one such $(\mathsf{pw}')^*$, send $(\mathsf{TestPwd}, sid, P', (\mathsf{pw}')^*)$ to $\mathcal{F}$; if there is no such $(\mathsf{pw}')^*$, send $(\mathsf{TestPwd}, sid, P', \perp)$ to $\mathcal{F}$.

      i. If $\mathcal{F}$ replies with "wrong guess", send $(\mathsf{NewKey}, sid, P', 0^n)$ to $\mathcal{F}$; sample $pk' \leftarrow \mathbb{G}$, compute $(c, \star) \leftarrow \mathsf{Encap}(pk')$, sample $\tau' \leftarrow \{0,1\}^{2n}$, and send (c, τ') from P' to P.

      ii. If $\mathcal{F}$ replies with "correct guess", and $(\mathsf{pw}')^*$ is defined in type (1) above, then r^* is already defined. Compute
      $$R' := H_1((\mathsf{pw}')^*, r^*), \quad pk' := T^*/R'.$$
      If $\mathcal{F}$ replies with "correct guess", and $(\mathsf{pw}')^*$ is defined in type (2) above, then $\mathcal{A}$ has queried $H_2((\mathsf{pw}')^*, T)$, and a record $\langle (\mathsf{pw}')^*, (pk')^*, \star \rangle$ must have been stored at that time (see the "RO queries" paragraph below). Compute
      $$pk' := (pk')^* \cdot (T^*/T).$$
      Either way, compute $(c, K') \leftarrow \mathsf{Encap}(pk')$, $SK' := H_{\mathsf{key}}(K')$, $\tau' := H_{\mathsf{tag}}((\mathsf{pw}')^*, pk', s^*, T^*, c, K')$, and send (c, τ') from P' to P.

Figure 6: UC simulator $\mathcal{S}$ for OEKE (part 1). Red text denotes difference from the simulator in [ABJ25]

P's session key

On (c^*, τ^*) from $\mathcal{F}$, simulate P's output as follows:

1. If $(s^*, T^*) = (s, T) \wedge (c^*, \tau^*) = (c, \tau')$, send $(\mathsf{NewKey}, sid, P, 0^n)$ to $\mathcal{F}$. // $\mathcal{A}$ is passive
2. Otherwise find $\mathcal{A}$'s $H_{\mathsf{tag}}(\mathsf{pw}^*, pk^*, s, T, c^*, K^*)$ query resulting in τ^*.
   (a) If there is no such query or more than one such query, send $(\mathsf{TestPwd}, sid, P, \perp)$ and then $(\mathsf{NewKey}, sid, P, 0^n)$ to $\mathcal{F}$.
   (b) If there is exactly one such query, retrieve record $\langle \mathsf{pw}^*, pk^*, sk^* \rangle$. If there is no such record, send $(\mathsf{TestPwd}, sid, P, \perp)$ and then $(\mathsf{NewKey}, sid, P, 0^n)$ to $\mathcal{F}$.
   (c) Compute $K := \mathsf{Decap}(sk^*, c^*)$. If $K^* \neq K$, send $(\mathsf{TestPwd}, sid, P, \perp)$ and then $(\mathsf{NewKey}, sid, P', 0^n)$ to $\mathcal{F}$.
   (d) Send $(\mathsf{TestPwd}, sid, P, \mathsf{pw}^*)$ to $\mathcal{F}$.
      i. If $\mathcal{F}$ replies with "wrong guess", send $(\mathsf{NewKey}, sid, P, 0^n)$ to $\mathcal{F}$.
      ii. If $\mathcal{F}$ replies with "correct guess", compute $SK := H_{\mathsf{key}}(K)$ and send $(\mathsf{NewKey}, sid, P, SK)$ to $\mathcal{F}$.

RO queries

Answer all of $\mathcal{A}$'s RO queries via lazy sampling, except for the following:
When (s, T) is sampled (see the "P-to-P' message" paragraph above), if $\mathcal{A}$ has already queried $H_2(\star, T)$, abort.
On $\mathcal{A}$'s $H_2(\mathsf{pw}^*, T)$ query (let the result be t^*), generate $(pk^*, sk^*) \leftarrow \mathsf{Gen}$ and compute

$$r^* := s \oplus t^*, \quad R^* := T/pk^*.$$

If $H_1(\mathsf{pw}^*, r^*)$ has already been defined and it is not R^*, abort. Otherwise program $H_1(\mathsf{pw}^*, r^*) := R^*$, and store $\langle \mathsf{pw}^*, pk^*, sk^* \rangle$.

Figure 7: UC simulator $\mathcal{S}$ for OEKE (part 2)

4.2 Game-Hop Argument

We now prove that for any PPT environment $\mathcal{Z}$, the simulator $\mathcal{S}$ in Section 4.1.2 generates an ideal-world view indistinguishable from $\mathcal{Z}$'s real-world view. The sequence of games starts from the real world (Game 0) and ends in the ideal world (Game 17). We assume that $\mathcal{A}$ makes up to $q_1, q_2, q_{\mathsf{key}}, q_{\mathsf{tag}}$ queries to ROs $H_1, H_2, H_{\mathsf{key}}, H_{\mathsf{tag}}$, respectively. By Lemma 1, we may additionally assume that the $(q_1 + 1)(q_2 + 1)$-UNI-ANO-RO property holds for the KEM scheme. For two adjacent games i and $i + 1$, let $\mathbf{Dist}_{\mathcal{Z}}^{i,i+1}$ be $\mathcal{Z}$'s distinguishing advantage between them.

A game-by-game comparison with the proof in [ABJ25, Section 5] is shown in *red*, with the parts that critically rely on certain items being included as inputs to H_{tag} *underlined*.

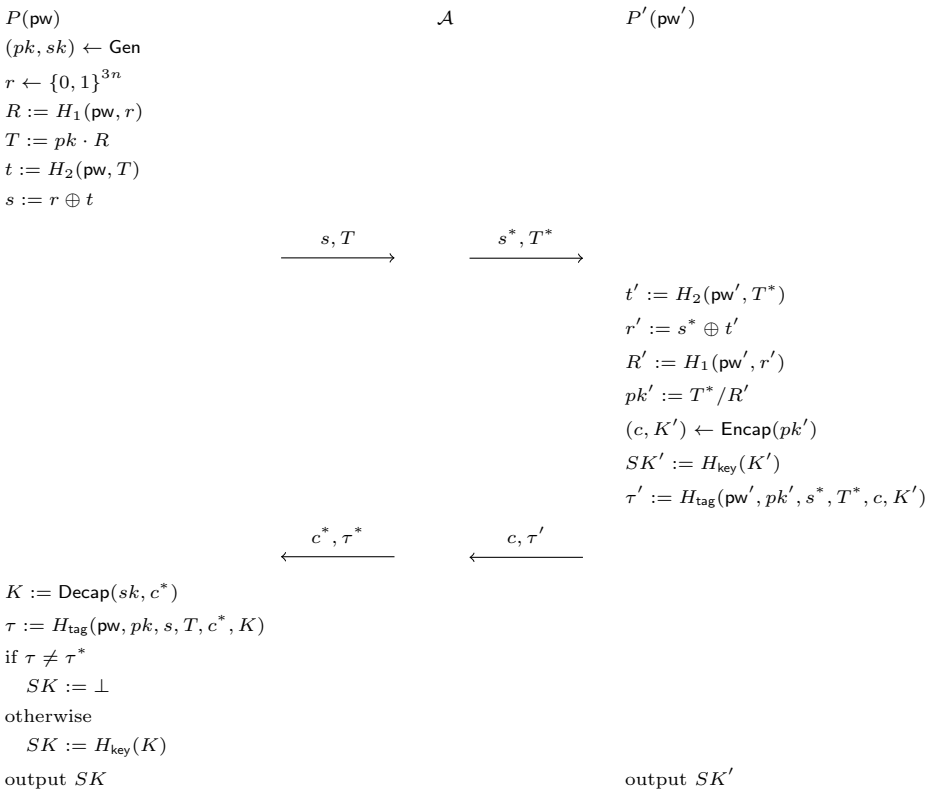

Figure 8: Game 0 (the real world)

Game 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 (s, T) to P';
- It is intercepted by the MitM adversary $\mathcal{A}$, who sends (s^*, T^*) (which may or may not be equal to (s, T)) to P';
- P' outputs session key SK' and sends (c, τ') to P;
- It is again intercepted by $\mathcal{A}$, who sends (c^*, τ^*) to P;
- P computes τ. If $\tau = \tau^*$ then P outputs an n-bit string SK, otherwise P outputs $\perp$.

(Note that there are three τ values: τ' computed and sent by P', τ^* sent by $\mathcal{A}$, and τ computed by P used in P's check.) A graphic illustration is shown in Figure 8.

This game corresponds to game G_1 in [ABJ25].

P and P''s passwords match and $\mathcal{A}$ is passive, except maybe modifying the tag.

Game 1: In the case where $\mathsf{pw} = \mathsf{pw}' \wedge (s^*, T^*) = (s, T) \wedge c^* = c$, do the following:

- If $\tau^* = \tau'$ (i.e., $\mathcal{A}$ is passive), set $SK := SK'$.
- If $\tau^* \neq \tau'$, set $SK := \perp$.

By correctness of the KEM scheme (cf. Section 3), in Game 0 $K = K'$, and thus $\tau = \tau'$, except with probability $\varepsilon_{\text{correctness}}$. This means that except with probability $\varepsilon_{\text{correctness}}$, P's check $\tau^* \stackrel{?}{=} \tau$ in Game 0 is equivalent to P's check $\tau^* \stackrel{?}{=} \tau'$ in Game 1. If the check passes then both games set $SK := H_{\mathsf{key}}(K)$, otherwise both games set $SK := \perp$. We have

$$\mathbf{Dist}^{0,1}_{\mathcal{Z}} \leq \varepsilon_{\text{correctness}}.$$

This game corresponds to games G_3–G_5 in [ABJ25]. We believe having a single game that handles both the $\tau^* = \tau'$ case and the $\tau^* \neq \tau'$ case is much cleaner.

P and P''s passwords match, and $\mathcal{A}$ forwards the P-to-P' message.

Game 2: In the case where $\neg(\mathsf{pw} = \mathsf{pw}' \wedge (s^*, T^*) = (s, T) \wedge c^* = c)$, find $\mathcal{A}$'s $H_{\mathsf{tag}}(\mathsf{pw}, pk, s, T, c^*, K^*)$ query resulting in τ^*. If there is none or more than one, set $SK := \perp$. If there is a unique such query, do the following:

- If $K^* = K$, set $SK := H_{\mathsf{key}}(K)$.
- If $K^* \neq K$, set $SK := \perp$.

Claim 1. *P' does not query $H_{\mathsf{tag}}(\mathsf{pw}, pk, s, T, c^*, K)$.*

Proof. P' makes a single H_{tag} query, on $(\mathsf{pw}', pk', s^*, T^*, c, K')$. Since $\neg(\mathsf{pw} = \mathsf{pw}' \wedge (s^*, T^*) = (s, T) \wedge c^* = c)$, of course P''s query is not $H_{\mathsf{tag}}(\mathsf{pw}, pk, s, T, c^*, K)$.[15] □

Going back to the game hop, Game 1 and Game 2 are identical unless one of the following happens:

1. $\mathcal{A}$ never makes an $H_{\mathsf{tag}}(\mathsf{pw}, pk, s, T, c^*, \star)$ query resulting in τ^*, yet P's check $\tau^* \stackrel{?}{=} H_{\mathsf{tag}}(\mathsf{pw}, pk, s, T, c^*, K)$ passes.
2. $\mathcal{A}$ makes more than one $H_{\mathsf{tag}}(\mathsf{pw}, pk, s, T, c^*, \star)$ query resulting in τ^*.
3. $\mathcal{A}$ makes an $H_{\mathsf{tag}}(\mathsf{pw}, pk, s, T, c^*, K^*)$ query (where $K^* \neq K$) resulting in τ^*, and P's $H_{\mathsf{tag}}(\mathsf{pw}, pk, s, T, c^*, K)$ query also results in τ^*.

[15] The proof of this claim is trivial, but we explicitly present it as a claim since its counterparts in some variants are much harder to prove.

In the first event, by Claim 1 P' does not query $H_{\mathsf{tag}}(\mathsf{pw}, pk, s, T, c^*, K)$. $\mathcal{A}$ itself also does not query $H_{\mathsf{tag}}(\mathsf{pw}, pk, s, T, c^*, K)$ (otherwise the result is τ^*). So $H_{\mathsf{tag}}(\mathsf{pw}, pk, s, T, c^*, K)$ is independent of $\mathcal{A}$'s view, and the probability that $\mathcal{A}$ sends $\tau^* = H_{\mathsf{tag}}(\mathsf{pw}, pk, s, T, c^*, K)$ is $1/2^{2n}$. The second and third events are collision in H_{tag} from either $\mathcal{A}$'s or P's queries, whose (combined) probability is upper-bounded by $q_{\mathsf{tag}}(q_{\mathsf{tag}}+1)/2^{2n+1}$. Overall,

$$\mathbf{Dist}^{1,2}_{\mathcal{Z}} \leq \frac{q_{\mathsf{tag}}(q_{\mathsf{tag}}+1)+2}{2^{2n+1}}.$$

This game corresponds to games G_6 and G_7 in [ABJ25], except that the H_{tag} collision cases (events (2) and (3) above) were ruled out in game G_2 in [ABJ25] (as aborting condition [A1]).

The proof of Claim 1 critically relies on pw, s, T, c being included in H_{tag} inputs.

Game 3: In the case where $\mathsf{pw} = \mathsf{pw}' \wedge (s^*, T^*) = (s, T)$, sample $SK', \tau' \leftarrow \{0,1\}^n$. ($c$ is still computed as before.)

In Game 2 P' computes $SK' := H_{\mathsf{key}}(K')$ and $\tau' := H_{\mathsf{tag}}(\mathsf{pw}', pk', s^*, T^*, c, K')$ $= H_{\mathsf{tag}}(\mathsf{pw}, pk, s, T, c, K')$. Let Query_1 be the event that $\mathcal{A}$ queries *either* $H_{\mathsf{key}}(K')$ *or* $H_{\mathsf{tag}}(\mathsf{pw}, pk, s, T, c, K')$; clearly Game 2 and Game 3 are identical unless Query_1 happens. We upper-bound $\Pr[\mathsf{Query}_1]$ via a reduction $\mathcal{R}_{\text{OW-1PCA}}$ to the OW-1PCA property of the KEM scheme:

Reduction $\mathcal{R}_{\text{OW-1PCA}}$: // Boxed text indicates the single PC oracle query; same for reductions below

Sample $i \leftarrow [q_{\mathsf{key}} + q_{\mathsf{tag}}]$ as a guess that $\mathcal{A}$'s i-th H_{key} or H_{tag} query triggers Query_1.
On input (pk, c),

1. Invoke $\mathcal{Z}$. On $(\mathsf{NewSession}, sid, P, P', \mathsf{pw}, \text{“initiator”})$ from $\mathcal{Z}$, run P's algorithm to compute (s, T), except that $\mathcal{R}_{\text{OW-1PCA}}$ uses its input pk instead of $(pk, \star) \leftarrow \mathsf{Gen}$. (Note that computing (s, T) does not require knowing sk.) Send (s, T) to $\mathcal{A}$.

2. On $(\mathsf{NewSession}, sid, P', P, \mathsf{pw}', \text{“respondent”})$ from $\mathcal{Z}$ and (s^*, T^*) from $\mathcal{A}$, if $\neg(\mathsf{pw} = \mathsf{pw}' \wedge (s^*, T^*) = (s, T))$, abort. Otherwise sample $SK' \leftarrow \{0,1\}^n, \tau' \leftarrow \{0,1\}^{2n}$, output SK' to $\mathcal{Z}$, and send (c, τ') to $\mathcal{A}$. (c is part of $\mathcal{R}_{\text{OW-1PCA}}$'s input.)

3. On (c^*, τ^*) from $\mathcal{A}$,

   (a) If $c^* = c$, do what's described in Game 1.

   (b) If $c^* \neq c$, find $\mathcal{A}$'s $H_{\mathsf{tag}}(\mathsf{pw}, pk, s, T, c^*, K^*)$ query resulting in τ^*. If there is none or more than one, output $SK := \perp$ to $\mathcal{Z}$. (This matches Game 2.)

(c) $\boxed{\text{Query } \mathsf{PC}_{sk}(c^*, K^*)}$. // This is a valid query since we do this step only if $c^* \neq c$
If the result is 1, output $SK := H_{\mathsf{key}}(K^*)$ to $\mathcal{Z}$.
If the result is 0, output $SK := \perp$ to $\mathcal{Z}$.

4. On $\mathcal{A}$'s i-th H_{key} or H_{tag} query, if it is in the form of $H_{\mathsf{key}}((K')^*)$ or $H_{\mathsf{tag}}(\mathsf{pw}, pk, s, T, c, (K')^*)$ for some $(K')^*$, output $(K')^*$; otherwise abort.

Assuming Query_1 happens, the only difference between Game 2/3 and $\mathcal{R}_{\text{OW-1PCA}}$'s simulation (until Query_1 happens) is that $\mathcal{R}_{\text{OW-1PCA}}$ uses its input pk on the P side, and then uses its input $c \leftarrow \mathsf{Encap}(pk)$ on the P' side. Since we assume $\mathsf{pw} = \mathsf{pw}' \wedge (s^*, T^*) = (s, T)$, in Game 2/3 P''s algorithm will recover $pk' = pk$, so there c is also computed as $\mathsf{Encap}(pk)$ and thus $\mathcal{R}_{\text{OW-1PCA}}$ simulates Game 2/3 perfectly. Furthermore, if $\mathcal{R}_{\text{OW-1PCA}}$'s guess is correct then it wins the OW-1PCA game. We have

$$\mathbf{Dist}^{2,3}_{\mathcal{Z}} \leq \Pr[\mathsf{Query}_1] \leq (q_{\mathsf{key}} + q_{\mathsf{tag}})\mathbf{Adv}^{\text{OW-1PCA}}_{\mathcal{R}_{\text{OW-1PCA}}}.$$

This game corresponds to game G_8 in [ABJ25]. However, there are two differences:

1. *[ABJ25] reduces to OW-PCA (where the KEM adversary can query its plaintext-checking oracle polynomially many times), rather than OW-1PCA. In this case, step 4 of the reduction above can query $\mathsf{PC}_{sk}(c, (K')^*)$ $q_{\mathsf{key}} + q_{\mathsf{tag}}$ times to check which of $\mathcal{A}$'s queries (if any) triggers Query_1, hence avoiding the $q_{\mathsf{key}} + q_{\mathsf{tag}}$ loss in its advantage.*

2. *The above also means that the plaintext-checking oracle in their OW-PCA game has to allow for $\mathsf{PC}_{sk}(c, \star)$ queries, whereas our reduction to OW-1PCA never makes a query in this form and thus the plaintext-checking oracle in our OW-1PCA game can disallow such queries.*

Game 4: Again in the case where $\mathsf{pw} = \mathsf{pw}' \wedge (s^*, T^*) = (s, T)$, generate P''s KEM public key $(pk', \star) \leftarrow \mathsf{Gen}$ and compute $(c, \star) \leftarrow \mathsf{Encap}(pk')$. (Note that K' is not used anywhere since Game 3 — in particular, P' does not use K' to compute SK' or τ' anymore — so we do not need to explicitly mention it.)

The difference between Game 3 and Game 4 is that in Game 3 both P and P' use the same pk, whereas in Game 4 P uses pk and P' uses an independent pk'. We construct a reduction $\mathcal{R}_{\text{ANO-1PCA}}$ to the ANO-1PCA property of the KEM scheme. This reduction, in fact, is identical to the reduction $\mathcal{R}_{\text{OW-1PCA}}$ to the OW-1PCA property in Game 3 except the last step; for completeness, we still present the full reduction and highlight the difference in blue:

Reduction $\mathcal{R}_{\text{ANO-1PCA}}$:

On input (pk, c),

1. Invoke $\mathcal{Z}$. On $(\mathsf{NewSession}, sid, P, P', \mathsf{pw}, \text{“initiator”})$ from $\mathcal{Z}$, run P's algorithm to compute (s, T), except that $\mathcal{R}_{\text{ANO-1PCA}}$ uses its input pk instead of $(pk, \star) \leftarrow \mathsf{Gen}$. Send (s, T) to $\mathcal{A}$.

2. On $(\mathsf{NewSession}, sid, P', P, \mathsf{pw}', \text{“respondent”})$ from $\mathcal{Z}$ and (s^*, T^*) from $\mathcal{A}$, if $\neg(\mathsf{pw} = \mathsf{pw}' \wedge (s^*, T^*) = (s, T))$, output 1 and halt. Otherwise sample $SK' \leftarrow \{0,1\}^n, \tau' \leftarrow \{0,1\}^{2n}$, output SK' to $\mathcal{Z}$, and send (c, τ') to $\mathcal{A}$. (c is part of $\mathcal{R}_{\text{ANO-PCA1}}$'s input.)

3. On (c^*, τ^*) from $\mathcal{A}$,

   (a) If $c^* = c$, do what's described in Game 1.

   (b) If $c^* \neq c$, find $\mathcal{A}$'s $H_{\mathsf{tag}}(\mathsf{pw}, pk, s, T, c^*, K^*)$ query resulting in τ^*. If there is none or more than one, output $SK := \perp$ to $\mathcal{Z}$.

   (c) $\boxed{\text{Query } \mathsf{PC}_{sk}(c^*, K^*)}$. // This is a valid query since we do this step only if $c^* \neq c$
   If the result is 1, output $SK := H_{\mathsf{key}}(K^*)$ to $\mathcal{Z}$.
   If the result is 0, output $SK := \perp$ to $\mathcal{Z}$.

4. When $\mathcal{Z}$ outputs bit b^*, copy $\mathcal{Z}$'s output.

$\mathcal{R}_{\text{ANO-1PCA}}$ uses its input pk on the P side and c on the P' side. If $\mathcal{R}_{\text{ANO-1PCA}}$'s challenge bit $b = 0$ then $c \leftarrow \mathsf{Encap}(pk)$, i.e., both P and P' use the same pk, so $\mathcal{R}_{\text{ANO-1PCA}}$ simulates Game 3; if $b = 1$ then $c \leftarrow \mathsf{Encap}(pk')$, i.e., P' uses an independently generated pk', so $\mathcal{R}_{\text{ANO-1PCA}}$ simulates Game 4. We have

$$\mathbf{Dist}^{3,4}_{\mathcal{Z}} \leq \mathbf{Adv}^{\text{ANO-1PCA}}_{\mathcal{R}_{\text{ANO-1PCA}}}.$$

This game corresponds to game G_9 in [ABJ25]. However, there are two differences:

1. *[ABJ25] reduces to ANO-PCA (where the KEM adversary can query its plaintext-checking oracle polynomially many times), rather than ANO-1PCA. Unlike in Game 3, here reducing to this stronger assumption does not provide any advantage.*

2. *The assumption [ABJ25] reduces to is even further stronger in that their KEM adversary's input is (pk, pk', c), whereas we do not give pk' to the KEM adversary. Note that the reduction does not need pk' at all (it only needs to send c which is an encapsulation of either pk or pk'), so the assumption in [ABJ25] is unnecessarily strong.*

P and P''s passwords do not match, and $\mathcal{A}$ forwards the P-to-P' message.

Game 5: After P sends (s, T) aimed at P', when $\mathcal{A}$ makes a fresh query $H_2(\mathsf{pw}^*, T)$ for $\mathsf{pw}^* \neq \mathsf{pw}$ (let the result be t^*), generate $(pk^*, sk^*) \leftarrow \mathsf{Gen}$, and compute

$$r^* := s \oplus t^*, \quad R^* := T/pk^*.$$

If $H_1(\mathsf{pw}^*, r^*)$ has already been defined and it is not R^*, abort. Otherwise program $H_1(\mathsf{pw}^*, r^*) := R^*$, and store $\langle \mathsf{pw}^*, pk^*, sk^* \rangle$.

Furthermore, in the case where $\mathsf{pw} \neq \mathsf{pw}' \wedge (s^*, T^*) = (s, T)$, when P' queries $H_2(\mathsf{pw}', T)$, use the method described above to generate pk', r', R' and program H_1. However, no record is stored in this case. (Note that the case where $\mathsf{pw} = \mathsf{pw}' \wedge (s^*, T^*) = (s, T)$ has already been addressed in Game 3 and Game 4, and P' does not need to make an H_2 query there.)[16]

In the analysis below, we combine $\mathcal{A}$'s $H_2(\mathsf{pw}^*, T)$ queries and P''s (possible) $H_2(\mathsf{pw}', T)$ query, and call all of them "$H_2(\mathsf{pw}^*, T)$". We first upper-bound the aborting probability. For every fresh $H_2(\mathsf{pw}^*, T)$ query, an independent $t^* \leftarrow \{0,1\}^{3n}$ value is sampled, which results in an independent $r^* \leftarrow \{0,1\}^{3n}$ (regardless of the distribution of s). As such, there are up to $q_2 + 1$ independent and uniformly random r^* values. The probability that one of $\mathcal{A}$'s q_1 $H_1(\mathsf{pw}^*, r^*)$ queries happens *before* the $H_2(\mathsf{pw}^*, T)$ query (when r^* is generated) is upper-bounded by $q_1(q_2 + 1)/2^{3n}$.[17]

Now suppose the aborting condition does not happen. The difference between Game 4 and Game 5 is that Game 4 samples $H_1(\mathsf{pw}^*, r^*)$ uniformly at random, whereas Game 5 defines it as T/pk^*. Note that

- For $H_2(\mathsf{pw}^*, T)$ queries where $\mathsf{pw}^* \neq \mathsf{pw}'$, sk^* (or even pk^*) is not used anywhere in the game;
- For the $H_2(\mathsf{pw}', T)$ query, we cannot say the above anymore, since pk' is used while computing c and K'. However, since $\mathsf{pw} \neq \mathsf{pw}'$, we know that $pk = T/H_1(\mathsf{pw}, r)$ is independent of $pk' = T/H_1(\mathsf{pw}', r')$, so sk' is still not used anywhere else in the game.

So, the distinguishing advantage between Game 4 and Game 5 can be reduced to the UNI-PK property of the KEM scheme. We only sketch the reduction $\mathcal{R}_{\text{UNI-PK}}$ here: $\mathcal{R}_{\text{UNI-PK}}$, on input pk^* (which is either an honestly generated public key or a uniformly random value), samples $i \leftarrow [q_2 + 1]$, and answers the (either $\mathcal{A}$'s or P''s) first $i - 1$ H_2 queries as in Game 4 and the last $q_2 - i + 1$ H_2 queries as in Game 5; for the i-th H_2 query, $\mathcal{R}_{\text{UNI-PK}}$ computes

$$r^* := s \oplus t^*, \quad R^* := T/pk^*$$

and programs $H_1(\mathsf{pw}^*, r^*) := R^*$. If this H_2 query is on (pw', T), $\mathcal{R}_{\text{UNI-PK}}$ also uses its input pk^* to compute c and K'. (Note that the $\langle \mathsf{pw}^*, pk^*, sk^* \rangle$ record is not actually used in Game 5, so $\mathcal{R}_{\text{UNI-PK}}$ does not need to store it.) $\mathcal{R}_{\text{UNI-PK}}$ simulates other parts of Game 4/5 honestly. Finally, $\mathcal{R}_{\text{UNI-PK}}$ copies $\mathcal{Z}$'s output bit.

[16] A slightly confusing case is when $s^* \neq s \wedge T^* = T$, causing P' to query $t' := H_2(\mathsf{pw}', T)$; and $\mathcal{A}$ also queries $H_2(\mathsf{pw}', T)$. In this case P''s $r' = s^* \oplus t'$ and $\mathcal{A}$'s $r^* = s \oplus t'$ are different, causing P''s values R', pk' and $\mathcal{A}$'s values R^*, pk^* to be independent of each other. Therefore, the game should run P''s algorithm as usual when P' queries $H_2(\mathsf{pw}', T)$ (which generates R', pk'), and use the new method described in Game 5 when $\mathcal{A}$ makes the same query (which generates R^*, pk^*).

[17] P also makes an H_1 query, but it is on (pw, r) where $\mathsf{pw} \neq \mathsf{pw}^*$ per the condition of this game (so it is impossible for this H_1 query to trigger an abortion).

A standard hybrid argument shows that (assuming the aborting condition does not happen) $\mathcal{Z}$'s distinguishing advantage between Game 4 and Game 5 is upper-bounded by $(q_2+1)\mathbf{Adv}^{\text{UNI-PK}}_{\mathcal{R}_{\text{UNI-PK}}}$.[18] Overall, we have

$$\mathbf{Dist}^{4,5}_{\mathcal{Z}} \leq (q_2+1)\mathbf{Adv}^{\text{UNI-PK}}_{\mathcal{R}_{\text{UNI-PK}}} + \frac{q_1(q_2+1)}{2^{3n}}.$$

This game corresponds to game G_{10} in [ABJ25]. However, there are two differences:

1. *[ABJ25] runs the procedure above to generate pk', r', R' and program H_1 as long as P' makes an $H_2(\mathsf{pw}', T)$ query — that is, including the case where $s^* \neq s \wedge T^* = T$. (If $T^* \neq T$ then of course P' does not make an $H_2(\mathsf{pw}', T)$ query and there is no change from the real world.) The reason for making this change is to prepare for Game 6 and Game 7 below, which simplifies P''s algorithm when $\mathsf{pw} \neq \mathsf{pw}' \wedge (s^*, T^*) = (s, T)$ (i.e., if we did not make the change in Game 5 first, the reductions in Game 6 and Game 7 would not go through); therefore, it is unnecessary (and potentially confusing) to make this change when $s^* \neq s \wedge T^* = T$.*

2. *The aborting case was ruled out in game G_2 in [ABJ25] (as aborting condition [A2b1]).*

Game 6: In the case where $\mathsf{pw} \neq \mathsf{pw}' \wedge (s^*, T^*) = (s, T)$, sample $SK', \tau' \leftarrow \{0,1\}^n$. ($c$ is still computed as before.)

In Game 5 P' computes $SK' := H_{\mathsf{key}}(K')$ and $\tau' := H_{\mathsf{tag}}(\mathsf{pw}', pk', s^*, T^*, c, K')$ $= H_{\mathsf{tag}}(\mathsf{pw}', pk', s, T, c, K')$. Let Query_2 be the event that $\mathcal{A}$ queries *either* $H_{\mathsf{key}}(K')$ *or* $H_{\mathsf{tag}}(\mathsf{pw}', pk', s, T, c, K')$; clearly Game 5 and Game 6 are identical unless Query_2 happens. We upper-bound $\Pr[\mathsf{Query}_2]$ via a reduction $\mathcal{R}_{\text{OW1}}$ to the OW property of the KEM scheme. This is somewhat similar to the reduction $\mathcal{R}_{\text{OW-1PCA}}$ to the OW-1PCA property in Game 3, and we highlight the differences in blue:

Reduction $\mathcal{R}_{\text{OW1}}$:

Sample $i \leftarrow [q_{\mathsf{key}} + q_{\mathsf{tag}}]$ as a guess that $\mathcal{A}$'s i-th H_{key} or H_{tag} query triggers Query_2, and $j \leftarrow [q_2+1]$ as a guess that the j-th H_2 query (by either $\mathcal{A}$ or P') is $H_2(\mathsf{pw}', T)$.
On input (pk', c),

1. Invoke $\mathcal{Z}$. On $(\mathsf{NewSession}, sid, P, P', \mathsf{pw}, \text{“initiator”})$ from $\mathcal{Z}$, run P's algorithm to compute (s, T). (This in particular defines sk.) Send (s, T) to $\mathcal{A}$.

[18] Note that here we cannot make q_2+1 separate reductions to the UNI-PK property, and must reduce to UNI-PK “in one blow”. Furthermore, the calculation of the reduction's advantage is more complicated than the normal case where the reduction wins as long as its guess is correct. This is similar to, say, why composing a PRG polynomial times yields another PRG; see, e.g., [BS23, Section 3.4.3] for details.

2. Whenever there is an $H_2(\star, T)$ query (by either $\mathcal{A}$ or $\mathcal{R}_{\mathrm{OW1}}$ itself on behalf of P' — see step 3 below), do what's described in Game 5. However, if this is the j-th query, use $\mathcal{R}_{\mathrm{OW1}}$'s input pk' instead of generating pk^* freshly.

3. On $(\mathsf{NewSession}, sid, P', P, \mathsf{pw}',$ "respondent") from $\mathcal{Z}$ and (s^*, T^*) from $\mathcal{A}$, if $\neg(\mathsf{pw} \neq \mathsf{pw}' \wedge (s^*, T^*) = (s, T))$, abort. Otherwise

   (a) If $\mathcal{A}$ has not queried $H_2(\mathsf{pw}', T)$, make this query on behalf of P'.

   (b) Sample $SK' \leftarrow \{0,1\}^n, \tau' \leftarrow \{0,1\}^{2n}$, output SK' to $\mathcal{Z}$, and send (c, τ') to $\mathcal{A}$. (c is part of $\mathcal{R}_{\mathrm{OW1}}$'s input.)

4. On (c^*, τ^*) from $\mathcal{A}$,

   (a) Find $\mathcal{A}$'s $H_{\mathsf{tag}}(\mathsf{pw}, pk, c^*, K^*)$ query resulting in τ^*. If there is none or more than one, output $SK := \perp$ to $\mathcal{Z}$. (This matches Game 2.)

   (b) Compute $K := \mathsf{Decap}(sk, c)$. ($sk$ is the KEM secret key generated in step 1.) // No need to query a plaintext-checking oracle since pk on the P side and pk' on the P' side are independent of each other, so the reduction can sample sk on its own
   If $K = K^*$, output $SK := H_{\mathsf{key}}(K)$ to $\mathcal{Z}$.
   If $K \neq K^*$, output $SK := \perp$ to $\mathcal{Z}$.

5. On $\mathcal{A}$'s i-th H_{key} or H_{tag} query, if it is in the form of $H_{\mathsf{key}}((K')^*)$ or $H_{\mathsf{tag}}(\mathsf{pw}', pk', s, T, c, (K')^*)$ for some $(K')^*$, output $(K')^*$; otherwise abort.

Assuming Query_2 happens and $\mathcal{R}_{\mathrm{OW1}}$'s guess of index j is correct, the only difference between Game 5/6 and $\mathcal{R}_{\mathrm{OW1}}$'s simulation (until Query_2 happens) is that $\mathcal{R}_{\mathrm{OW1}}$ uses its input pk' to program $H_1(\mathsf{pw}', r')$, and then uses its input $(c, \star) \leftarrow \mathsf{Encap}(pk')$ as the P'-to-P message. Since c is also computed as $\mathsf{Encap}(pk')$ in Game 5/6, $\mathcal{R}_{\mathrm{OW1}}$ simulates Game 5/6 perfectly. Furthermore, if $\mathcal{R}_{\mathrm{OW1}}$'s guess of index i is correct then it wins the OW game. We have

$$\mathbf{Dist}_{\mathcal{Z}}^{5,6} \leq \Pr[\mathsf{Query}_2] \leq (q_2 + 1)(q_{\mathsf{key}} + q_{\mathsf{tag}})\mathbf{Adv}^{\mathrm{OW}}_{\mathcal{R}_{\mathrm{OW1}}}.$$

Remark 9. It is very subtle why the reduction needs to guess the index j and lose a factor of $q_2 + 1$ (and we applaud [ABJ25] for getting it right). The reason is as follows. Consider the following environment $\mathcal{Z}$:

1. Initiate P's session on password pw and receive (s, T).

2. Instruct $\mathcal{A}$ to make a large number of $H_2(\mathsf{pw}^*, T)$ queries (let the result be t^*), one of which is $H_2(\mathsf{pw}', T)$ for a special value pw'.

3. For each of the H_2 queries in step 2,

$$\text{compute } r^* := s \oplus t^*, \quad \text{query } H_1(\mathsf{pw}^*, r^*).$$

4. Initiate P''s session on password pw' and instruct $\mathcal{A}$ to forward (s, T) to P'.

To simulate Game 5/6 correctly, $\mathcal{R}_{\mathrm{OW1}}$ must use its input pk' to program $H_1(\mathsf{pw}', r') := T/pk'$ in step 3. The problem is that $\mathcal{R}_{\mathrm{OW1}}$ *does not know* pw' *until step 4, so it has no idea which* $H_1(\mathsf{pw}', r')$ *query in step 3 needs to be programmed using* pk'! The only way to get around is to make a guess over all $H_2(\star, T)$ queries, which incurs a loss up to $q_2 + 1$. (If we require $\mathcal{Z}$ to always initiate P and P''s sessions simultaneously, this loss can be avoided since in step 3 $\mathcal{R}_{\mathrm{OW1}}$ must have already learned pw'. However, the UC PAKE functionality does not place this restriction on $\mathcal{Z}$.)

This game corresponds to game G_{11} *in [ABJ25], except that they reduce to OW-PCA and avois the* $q_{\mathsf{key}} + q_{\mathsf{tag}}$ *loss (cf. the comment after Game 3).*

Game 7: Again in the case where $\mathsf{pw} \neq \mathsf{pw}' \wedge (s^*, T^*) = (s, T)$, recall Game 6 does the following on the P' side: Generate $(pk', \star) \leftarrow \mathsf{Gen}$, query $t' := H_2(\mathsf{pw}', T)$, and compute

$$r' := s \oplus t', \quad R' := T/pk'.$$

((s, T) are the honest P-to-P' message.) If $H_1(\mathsf{pw}', r')$ has already been defined and it is not R', abort. Otherwise program $H_1(\mathsf{pw}', r') := R'$, <u>compute $(c, \star) \leftarrow \mathsf{Encap}(pk')$</u>, and sample $SK' \leftarrow \{0,1\}^n, \tau' \leftarrow \{0,1\}^{2n}$. Finally, output SK' and send (c, τ') to P. (Note that K' is not used anywhere since Game 6 — in particular, P' does not use K' to compute SK' or τ' anymore — so we do not need to explicitly mention it.) In this game, we independently generate $(pk'', \star) \leftarrow \mathsf{Gen}$ and compute $(c, \star) \leftarrow \mathsf{Encap}(pk'')$ (replacing the <u>underlined</u> expression).

The difference between Game 6 and Game 7 is that in Game 6 c is computed from

$$pk' = \frac{T}{H_1(\mathsf{pw}', s \oplus H_2(\mathsf{pw}', T))}$$

which $\mathcal{A}$ can compute, whereas in Game 7 c is computed from an independently generated pk''. We construct a reduction $\mathcal{R}_{\mathrm{ANO}}$ to the ANO property of the KEM scheme. This reduction, in fact, is identical to the reduction $\mathcal{R}_{\mathrm{OW1}}$ to the OW property in Game 6 except the last step; for completeness, we still present the full reduction and highlight the difference in blue:

<u>Reduction $\mathcal{R}_{\mathrm{ANO}}$</u>:

Sample $j \leftarrow [q_2 + 1]$ as a guess that the j-th H_2 query (by either $\mathcal{A}$ or P') is $H_2(\mathsf{pw}', T)$.
On input (pk', c),

1. Invoke $\mathcal{Z}$. On $(\mathsf{NewSession}, sid, P, P', \mathsf{pw}, \text{“initiator”})$ from $\mathcal{Z}$, run P's algorithm to compute (s, T). (This in particular defines sk.) Send (s, T) to $\mathcal{A}$.

2. Whenever there is an $H_2(\star, T)$ query (by either $\mathcal{A}$ or $\mathcal{R}_{\mathrm{ANO1}}$ itself on behalf of P' — see step 3 below), do what's described in Game 5. However, if this is the j-th query, use $\mathcal{R}_{\mathrm{ANO}}$'s input pk' instead of generating pk^* freshly.

3. On $(\mathsf{NewSession}, sid, P', P, \mathsf{pw}',$ "respondent") from $\mathcal{Z}$ and (s^*, T^*) from $\mathcal{A}$, if $\neg(\mathsf{pw} \neq \mathsf{pw}' \wedge (s^*, T^*) = (s, T))$, output 1 and halt. Otherwise
   (a) If $\mathcal{A}$ has not queried $H_2(\mathsf{pw}', T)$, make this query on behalf of P'.
   (b) Sample $SK' \leftarrow \{0,1\}^n, \tau' \leftarrow \{0,1\}^{2n}$, output SK' to $\mathcal{Z}$, and send (c, τ') to $\mathcal{A}$. (c is part of $\mathcal{R}_{\mathrm{ANO}}$'s input.)
4. On (c^*, τ^*) from $\mathcal{A}$,
   (a) Find $\mathcal{A}$'s $H_{\mathsf{tag}}(\mathsf{pw}, pk, s, T, c^*, K^*)$ query resulting in τ^*. If there is none or more than one, output $SK := \perp$ to $\mathcal{Z}$.
   (b) Compute $K := \mathsf{Decap}(sk, c^*)$. ($sk$ is the KEM secret key generated in step 1.) // No need to query a plaintext-checking oracle since pk on the P side and pk' on the P' side are independent of each other, so the reduction can sample sk on its own
   If $K = K^*$, output $SK := H_{\mathsf{key}}(K)$ to $\mathcal{Z}$.
   If $K \neq K^*$, output $SK := \perp$ to $\mathcal{Z}$.
5. When $\mathcal{Z}$ outputs bit b^*, copy $\mathcal{Z}$'s output.

Assume $\mathcal{R}_{\mathrm{ANO}}$'s guess of index j is correct. If $b = 0$ then $c \leftarrow \mathsf{Encap}(pk')$, i.e., the computation of R' and c uses the same pk', so $\mathcal{R}_{\mathrm{ANO}}$ simulates Game 6; if $b = 1$ then $c \leftarrow \mathsf{Encap}(pk'')$, i.e., the computation of c uses an independently generated pk'', so $\mathcal{R}_{\mathrm{ANO}}$ simulates Game 7. We have

$$\mathbf{Dist}_{\mathcal{Z}}^{6,7} \leq (q_2 + 1)\mathbf{Adv}_{\mathcal{R}_{\mathrm{ANO}}}^{\mathrm{ANO}}.$$

Game 8: Again in the case where $\mathsf{pw} \neq \mathsf{pw}' \wedge (s^*, T^*) = (s, T)$, we outright skip the computation of t', r', R' and the programming of H_1 (the brown text in Game 7). (Since pk' is not generated anymore, we can rename pk'' — the KEM public key from which c is computed — to pk'.)

Note that the aborting condition, which was originally introduced in Game 5, is removed (for the specific $H_2(\mathsf{pw}', T)$ query by P'). By an analysis similar to that in Game 5, the aborting probability in Game 7 is upper-bounded by $q_1/2^{3n}$. If the aborting condition is not met, the brown text is independent of the rest of the game, so removing it does not affect the distribution of $\mathcal{Z}$'s output. We have

$$\mathbf{Dist}_{\mathcal{Z}}^{7,8} \leq \frac{q_1}{2^{3n}}.$$

The two games above combined correspond to game G_{12} in [ABJ25]. [ABJ25] does not have the intermediate Game 7 in our proof, and outright removes the brown text from what's our Game 6. We believe it is clearer to include our Game 7, where H_2 is still queried and H_1 is still programmed: in this way it is easier to see why the reduction to ANO simulates the games correctly. Also, [ABJ25] reduces to ANO-PCA,[19] although it also says that the plaintext-checking oracle is not needed.

[19] [ABJ25] actually mentions ANO-CPA, which appears to be a typo.

Computation of the P-to-P' message.

Game 9: Note that up to now we have not changed P's algorithm before sending (s, T) (in particular, Game 5 dealt with H_2 queries by $\mathcal{A}$ and P', but not P). In this game we modify it as follows: On input $(\mathsf{NewSession}, sid, P, P', \mathsf{pw}, \text{“initiator”})$, sample $s \leftarrow \{0,1\}^{3n}, T \leftarrow \mathbb{G}$. If $\mathcal{A}$ has queried $H_2(\star, T)$, abort. Otherwise query $t := H_2(\mathsf{pw}, T)$, generate $(pk, \star) \leftarrow \mathsf{Gen}$, and compute

$$r := s \oplus t, \quad R := T/pk.$$

If $H_1(\mathsf{pw}, r)$ has already been defined and it is not R, also abort. Otherwise program $H_1(\mathsf{pw}, r) := R$ and store $\langle \mathsf{pw}, pk, sk \rangle$.

We first upper-bound the aborting probability. The probability that one of $\mathcal{A}$'s H_2 query inputs (before T is sampled) happens to contain T is upper-bounded by $q_2/|\mathbb{G}|$. By an analysis similar to that in Game 5 and Game 8, the probability of the second aborting event is upper-bounded by $q_1/2^{3n}$.

Now suppose neither of the aborting events happens. The following equations hold in both Game 8 and Game 9:

$$r = s \oplus H_2(\mathsf{pw}, T), \quad T = R \cdot pk.$$

Here, pk is generated by Gen, and $H_2(\mathsf{pw}, T)$ is fixed by an H_2 query once T is fixed. The four remaining variables r, s, R, T are constrained by two equations and have $4 - 2 = 2$ degrees of freedom: if we sample $r \leftarrow \{0,1\}^{3n}, R \leftarrow \mathbb{G}$ then we get Game 8, and if we sample $s \leftarrow \{0,1\}^{3n}, T \leftarrow \mathbb{G}$ then we get Game 9. Thus, Game 8 and Game 9 are identical. We have

$$\mathbf{Dist}^{8,9}_{\mathcal{Z}} \leq \frac{q_1}{2^{3n}} + \frac{q_2}{|\mathbb{G}|}.$$

The game above corresponds to game G_{13} in [ABJ25], with one difference: our game aborts if $\mathcal{A}$ has queried $H_2(\mathsf{pw}^*, T)$ for any pw^* when T is sampled, whereas the game in [ABJ25] aborts only if $\mathcal{A}$ has queried $H_2(\mathsf{pw}, T)$. Note that the simulator does not know pw, so it cannot tell if $\mathcal{A}$ has queried $H_2(\mathsf{pw}, T)$ and instead has to abort as long as $\mathcal{A}$ has queried $H_2(\star, T)$. (The simulator in [ABJ25] is correct, but its game hops do not match the ideal world regarding which H_2 queries will trigger an abortion.) Also, we believe our argument under this game is much more concise and cleaner (our presentation is, in spirit, similar to [JRX25, Lemma 4.5, Hybrid 2]).

Game 10: When P's session begins, halt after checking if $H_2(\star, T)$ has been defined (i.e., before the brown text in Game 9). Then when P receives (c^*, τ^*), if $\mathsf{pw} = \mathsf{pw}' \wedge (s^*, T^*) = (s, T) \wedge c^* = c$, do what's described in Game 1. Otherwise,

- If $\mathcal{A}$ has now queried $H_2(\mathsf{pw}, T)$, there must be a corresponding record $\langle \mathsf{pw}, pk, sk \rangle$ (see Game 5).
- If $\mathcal{A}$ still has not queried $H_2(\mathsf{pw}, T)$, run the brown text in Game 9.

Either way, pk, sk are defined and the game can proceed as usual.

The only difference between Game 9 and Game 10 lies in the fact that when we move the brown text around, the game might abort — as part of the brown text — at different times. Since the aborting event happens with probability at most $q_1/2^{3n}$, we have

$$\mathbf{Dist}_{\mathcal{Z}}^{9,10} \leq \frac{q_1}{2^{3n}}.$$

The game above corresponds to game G_{14} in [ABJ25]. Note that we have a $q_1/2^{3n}$ loss in both Game 9 and Game 10; if we remove Game 9 and directly jump to Game 10 from Game 8, this loss does not need to be repeated (since Game 8 and Game 10 are identical unless one of the two aborting events happens). We (as well as [ABJ25]) choose to include Game 9 for clarity.

Intermission. Let's pause a bit and review the changes we have made so far:

1. *P-to-P' message:* Game 10 outright samples (s, T) at random. [changed in Game 9]

2. *P''s session key and P'-to-P message:* On (s^*, T^*) from $\mathcal{A}$ to P', if $(s^*, T^*) = (s, T)$, Game 10 outright samples SK', τ' at random. Regarding c, Game 10 generates $(pk', \star) \leftarrow \mathsf{Gen}$ and computes $(c, \star) \leftarrow \mathsf{Encap}(pk')$. (If $(s^*, T^*) \neq (s, T)$ then there is no change from the real world.) [changed in Games 3, 4, 6, 7, 8]

3. *P's session key:* On (c^*, τ^*) from $\mathcal{A}$ to P, we have made the following changes:

   (a) If $\mathsf{pw} = \mathsf{pw}' \wedge (s^*, T^*) = (s, T) \wedge c^* = c$, Game 10 outright sets $SK := SK'$ if $\tau^* = \tau'$ and $SK := \bot$ otherwise. [changed in Game 1]

   (b) Otherwise if $\mathcal{A}$ makes no $H_{\mathsf{tag}}(\mathsf{pw}, pk, s, T, c^*, K^*)$ query resulting in τ^*, or more than one such query, Game 10 outright sets $SK := \bot$. [changed in Game 2]

   (c) Otherwise if sk has not been defined yet, Game 10 queries $H_2(\mathsf{pw}, T)$ on behalf of P, which generates sk (see step 4 below). Then Game 10 computes $K := \mathsf{Decap}(sk, c^*)$, and sets $SK := H_{\mathsf{key}}(K)$ if $K^* = K$ and $SK := \bot$ otherwise. [changed in Game 2]

4. *H_2 queries:* When either $\mathcal{A}$ or P makes a fresh query $H_2(\mathsf{pw}^*, T)$ (let the result be t^*), Game 10 runs the following procedure: Generate $(pk^*, sk^*) \leftarrow \mathsf{Gen}$, and compute

   $$r^* := s \oplus t^*, \quad R^* := T/pk^*.$$

   If $H_1(\mathsf{pw}^*, r^*)$ has already been defined and it is not R^*, abort. Otherwise program $H_1(\mathsf{pw}^*, r^*) := R^*$. If the H_2 query is made by $\mathcal{A}$, also store $\langle \mathsf{pw}^*, pk^*, sk^* \rangle$. [changed in Games 5, 10]

See Figure 9 for a summary of Game 10.

1 :	$P(\mathsf{pw})$		$\mathcal{A}$	$P'(\mathsf{pw}')$
2 :	$s \leftarrow \{0,1\}^{3n}; T \leftarrow \mathbb{G}$			
3 :		$\xrightarrow{s, T}$	$\xrightarrow{s^*, T^*}$	
4 :				if $(s^*, T^*) = (s, T)$
5 :				$SK' \leftarrow \{0,1\}^n, \tau' \leftarrow \{0,1\}^{2n}$
6 :				$(pk', \star) \leftarrow \mathsf{Gen}$
7 :				$(c, \star) \leftarrow \mathsf{Encap}(pk')$
8 :				otherwise
9 :				$t' := H_2(\mathsf{pw}', T^*)$
10 :				$r' := s^* \oplus t'$
11 :				$R' := H_1(\mathsf{pw}', r')$
12 :				$pk' := T^*/R'$
13 :				$(c, K') \leftarrow \mathsf{Encap}(pk')$
14 :				$SK' := H_{\mathsf{key}}(K')$
15 :				$\tau' := H_{\mathsf{tag}}(\mathsf{pw}', pk', s^*, T^*, c, K')$
16 :		$\xleftarrow{c^*, \tau^*}$	$\xleftarrow{c, \tau'}$	
17 :	if $\mathsf{pw} = \mathsf{pw}' \wedge (s^*, T^*) = (s, T) \wedge c^* = c$			
18 :	if $\tau^* = \tau$			
19 :	$SK := SK'$			
20 :	otherwise			
21 :	$SK := \perp$			
22 :	otherwise			
23 :	retrieve $\langle \mathsf{pw}, pk, sk \rangle$ record			
24 :	if none			
25 :	$t := H_2(\mathsf{pw}, T)$			
26 :	$(pk, sk) \leftarrow \mathsf{Gen}$			
27 :	$r := s \oplus t$			
28 :	$R := T/pk$			
29 :	$H_1(\mathsf{pw}, r) := R$			
30 :	store $\langle \mathsf{pw}, pk, sk \rangle$			
31 :	find $\mathcal{A}$'s $H_{\mathsf{tag}}(\mathsf{pw}, pk, s, T, c^*, K^*) = \tau^*$ query			
32 :	if none or more than one			
33 :	$SK := \perp$			
34 :	$K := \mathsf{Decap}(sk, c^*)$			
35 :	if $K^* = K$			
36 :	$SK := H_{\mathsf{key}}(K)$			
37 :	otherwise			
38 :	$SK := \perp$			
39 :	output SK			output SK'
40 :	on $\mathcal{A}$'s fresh $H_2(\mathsf{pw}^*, T) =: t^*$ query:			
41 :	if there is no $\langle \mathsf{pw}^*, \star, \star \rangle$ record			
42 :	$(pk^*, \star) \leftarrow \mathsf{Gen}$			
43 :	$r^* := s \oplus t^*$			
44 :	$R^* := T/pk^*$			
45 :	$H_1(\mathsf{pw}^*, r^*) := R^*$			
46 :	store $\langle \mathsf{pw}^*, pk^*, sk^* \rangle$			

Figure 9: Game 10 (with various aborting conditions omitted)

Password extraction from P'-to-P message.

Game 11: In the case where $\neg(\mathsf{pw} = \mathsf{pw}' \wedge (s^*, T^*) = (s, T) \wedge c^* = c)$, if there is no $\langle \mathsf{pw}, pk, sk \rangle$ record, set $SK := \perp$ (i.e., replace the brown text in Figure 9 with "$SK := \perp$").

Game 10 and Game 11 differ only if $\mathcal{A}$ does not make an $H_2(\mathsf{pw}, T)$ query, but makes an $H_{\mathsf{tag}}(\mathsf{pw}, pk, s, T, c^*, K^*)$ query resulting in τ^* (for c^*, K^*, τ^* of $\mathcal{A}$'s choice and pk generated in the brown text). This in particular requires $\mathcal{A}$ to know pk, which is freshly generated by Gen. Therefore, the probability that $\mathcal{A}$ can predict pk is upper-bounded by p_{guess}. We have

$$\mathbf{Dist}_{\mathcal{Z}}^{10,11} \leq p_{\text{guess}}.$$

The game above corresponds to game G_{15} in [ABJ25]. [ABJ25] does not include the condition $\neg(\mathsf{pw} = \mathsf{pw}' \wedge (s^*, T^*) = (s, T) \wedge c^* = c)$, i.e., it appears that this game rewrites what our Game 1 does, which is incorrect.

[ABJ25] uses the notation $H_\infty(pk : (pk, \star) \leftarrow \mathsf{Gen})$ instead of p_{guess}. By default this is the min-entropy of pk, namely $-\log p_{\text{guess}}$ (which is a value greater than 1). We believe $1/2^{H_\infty(pk)}$, or simply p_{guess}, is a better notation.

The argument in this game critically relies on pk being included in H_{tag} inputs.

Game 12: In the case where $\mathsf{pw} = \mathsf{pw}' \wedge (s^*, T^*) = (s, T) \wedge c^* = c \wedge \tau^* \neq \tau'$, do what's in the $\neg(\mathsf{pw} = \mathsf{pw}' \wedge (s^*, T^*) = (s, T) \wedge c^* = c)$ case (i.e., replace line 21 of Figure 9 with lines 23–38).

In Game 11 SK is outright set to $\perp$, whereas here it is treated the same as the "normal" case. Let Query_3 be the event that $\mathcal{A}$ queries $H_{\mathsf{tag}}(\mathsf{pw}, pk, s, T, c, K)$, where pk is the KEM public key generated on $\mathcal{A}$'s $H_2(\mathsf{pw}, T)$ query; Game 11 and Game 12 are identical unless Query_3 happens. To trigger Query_3, $\mathcal{A}$ must know K, which suggests a reduction $\mathcal{R}_{\text{OW2}}$ to the OW property of the KEM scheme. This reduction is simple, so we only provide a sketch: $\mathcal{R}_{\text{OW2}}$, on input (pk, c), invokes $\mathcal{Z}$ and simulates (s, T) and the P' side honestly. (Note that P' uses a freshly generated pk' independent of the P side.) When $\mathcal{A}$ queries $H_2(\mathsf{pw}, T)$, $\mathcal{R}_{\text{OW2}}$ uses its input pk in lieu of $(pk, \star) \leftarrow \mathsf{Gen}$. Along the way, as long as $\mathcal{R}_{\text{OW2}}$ detects that $\mathsf{pw} = \mathsf{pw}' \wedge (s^*, T^*) = (s, T) \wedge c^* = c \wedge \tau^* \neq \tau'$ does not happen, it aborts. Finally, $\mathcal{R}_{\text{OW2}}$ finds $\mathcal{A}$'s (unique) $H_{\mathsf{tag}}(\mathsf{pw}, pk, s, T, c, K^*)$ query resulting in τ^*, and outputs K^*. We have

$$\mathbf{Dist}_{\mathcal{Z}}^{11,12} \leq \Pr[\mathsf{Query}_3] \leq \mathbf{Adv}^{\text{OW}}_{\mathcal{R}_{\text{OW2}}}.$$

The game above is missing in [ABJ25]. Note that in the ideal world, the simulator cannot tell if $\mathsf{pw} = \mathsf{pw}'$, so when it sees $(s^*, T^*) = (s, T) \wedge c^* = c \wedge \tau^* \neq \tau'$, it has to always try to extract a password guess; hence, the game above has to be here. (In the $\mathsf{pw} = \mathsf{pw}' \wedge (s^*, T^*) = (s, T) \wedge c^* = c \wedge \tau^* \neq \tau'$ case, we first outright set $SK := \perp$ in Game 1 and then "walk back" in Game 12, since the change in Game 1 is necessary for the reductions in Games 3 and 4 to go through.)

Game 13: In the case where $\neg(\mathsf{pw} = \mathsf{pw}' \wedge (s^*, T^*) = (s, T) \wedge (c^*, \tau^*) = (c, \tau'))$, find $\mathcal{A}$'s $H_{\mathsf{tag}}(\mathsf{pw}^*, pk^*, s, T, c^*, K^*)$ query resulting in τ^*. (The difference is that Game 12 only goes over all of $\mathcal{A}$'s queries in the form of $H_{\mathsf{tag}}(\mathsf{pw}, pk, s, T, c^*, \star)$, whereas this game goes over all of $\mathcal{A}$'s queries in the form of $H_{\mathsf{tag}}(\star, \star, s, T, c^*, \star)$.) If there is none or more than one, set $SK := \perp$. Otherwise there is a unique such $H_{\mathsf{tag}}(\mathsf{pw}^*, pk^*, s, T, c^*, K^*)$ query, and

- If $\mathsf{pw}^* \neq \mathsf{pw}$, set $SK := \perp$.
- If $\mathsf{pw}^* = \mathsf{pw}$, there is no change from Game 12. That is, the game runs the following procedure:
  1. Retrieve record $\langle \mathsf{pw}, pk, sk \rangle$. If there is no such record, or if $pk^* \neq pk$, set $SK := \perp$.[20]
  2. Compute $K := \mathsf{Decap}(sk, c^*)$. If $K^* = K$, set $SK := H_{\mathsf{key}}(K)$; otherwise set $SK := \perp$.

If $\mathcal{A}$ makes no $H_{\mathsf{tag}}(\star, \star, s, T, c^*, \star)$ query resulting in τ^*, then in particular $\mathcal{A}$ makes no $H_{\mathsf{tag}}(\mathsf{pw}, pk, s, T, c^*, \star)$ query resulting in τ^*, so both Game 12 and Game 13 set $SK := \perp$. The probability that $\mathcal{A}$ makes two different $H_{\mathsf{tag}}(\star, \star, s, T, c^*, \star)$ queries both resulting in τ^* is upper-bounded by $q_{\mathsf{tag}}(q_{\mathsf{tag}} - 1)/2^{2n+1}$. Now suppose $\mathcal{A}$ makes a unique $H_{\mathsf{tag}}(\mathsf{pw}^*, pk^*, s, T, c^*, K^*)$ query resulting in τ^*:

- If $\mathsf{pw}^* \neq \mathsf{pw}$, then $\mathcal{A}$ does not make a $H_{\mathsf{tag}}(\mathsf{pw}, pk, s, T, c^*, \star)$ query resulting in τ^* (because there is only one $H_{\mathsf{tag}}(\star, \star, s, T, c^*, \star)$ query resulting in τ^*, and it is on pw^*), so both Game 12 and Game 13 set $SK := \perp$.
- If $\mathsf{pw}^* = \mathsf{pw}$, then there is no difference between Game 12 and Game 13.

We conclude that

$$\mathbf{Dist}_{\mathcal{Z}}^{12,13} \leq \frac{q_{\mathsf{tag}}(q_{\mathsf{tag}} - 1)}{2^{2n+1}}.$$

The game above is also missing in [ABJ25]. The H_{tag} collision case was ruled out in game G_2 in [ABJ25] (as aborting condition [A1]);[21] *however, conceptually speaking there still needs to be such a bridge game, to make sure that the game hops match the ideal world at the end.*

The definition of this game, together with its argument, critically rely on pw being included in H_{tag} inputs.

[20] The rule "If there is no $\langle \mathsf{pw}, pk, sk \rangle$ record, set $SK := \perp$" was introduced in Game 11. We check if $pk^* = pk$ because in Game 12 pk in the record (line 23 of Figure 9) and in $\mathcal{A}$'s H_{tag} query (line 31 of Figure 9) have to match each other; in other words, if $\mathcal{A}$'s H_{tag} query is on $(\mathsf{pw}, pk^*, s, T, c^*, \star)$ for some $pk^* \neq pk$, then Game 12 sets $SK := \perp$. Therefore, here we need to explicitly perform the $pk^* \stackrel{?}{=} pk$ check in order to be consistent with Game 12. In other OEKE variants where pk is removed from H_{tag} inputs, this check needs to be removed as well (since pk^* is not well-defined anymore).

[21] In our game hops, we ruled out H_{tag} collisions for different K^* values in Game 2, and then additionally rule out H_{tag} collisions for different $(\mathsf{pw}^*, pk^*, K^*)$ tuples in Game 13. We could have done both in Game 2 and avoided the additional loss here; however, we believe the current approach is clearer.

Game 14: In the case where $\mathsf{pw} \neq \mathsf{pw}' \wedge (s^*, T^*) = (s, T) \wedge (c^*, \tau^*) = (c, \tau')$, set $SK := \perp$.

In this case $\tau^* = \tau'$ is uniformly random and independent of the rest of the game, so the probability that $\mathcal{A}$ makes an H_{tag} query resulting in τ^* is at most $q_{\mathsf{tag}}/2^{2n}$. If $\mathcal{A}$ does not make such a query, then both Game 13 and Game 14 set $SK := \perp$. We have

$$\mathbf{Dist}_{\mathcal{Z}}^{13,14} \leq \frac{q_{\mathsf{tag}}}{2^{2n}}.$$

The game above is also missing in [ABJ25]. Again in the ideal world, the simulator cannot tell if $\mathsf{pw} = \mathsf{pw}'$, so when it sees $\mathcal{A}$ is passive, it has to let the PAKE functionality check if $\mathsf{pw} = \mathsf{pw}'$ and set SK to be equal to SK' or $\perp$ accordingly; hence, the game above has to be here.

Password extraction from P-to-P' message.

Game 15: In the case where $(s^*, T^*) \neq (s, T)$, find $\mathcal{A}$'s $H_2(\mathsf{pw}^*, T^*)$ query[22] (let the result be t^*) that satisfies *either* of the following:

1. $\mathcal{A}$ queried $H_1(\mathsf{pw}^*, r^*)$ *before* the $H_2(\mathsf{pw}^*, T^*)$ query, where $r^* = s^* \oplus t^*$.
2. $\mathcal{A}$ also queried $H_2(\mathsf{pw}^*, T)$, and
$$s \oplus s^* = H_2(\mathsf{pw}^*, T) \oplus t^*.$$

(This $H_2(\mathsf{pw}^*, T^*)$ query is what's called the "anchor query" in [JRX25].) If there is more than one such query, abort. (Intuitively pw^* is $\mathcal{A}$'s password guess contained in (s^*, T^*).)

The aborting condition is met only in the following three cases:

- There is a collision in type (1) above, i.e., there are $r^*, t^*, (r^*)', (t^*)'$ such that
$$r^* \oplus t^* = (r^*)' \oplus (t^*)'$$
(so $\mathcal{A}$ can set s^* to be this value). Note that all four values in the equation are determined by two (different) H_1 queries and one H_2 query, and for each set of these queries, there is a $1/2^{3n}$ chance that the equation will hold. Therefore, the probability of this case is upper-bounded by $q_1(q_1 - 1)q_2/2^{3n}$.
- There is a collision in type (2) above, i.e., there are $\mathsf{pw}^*, t^*, (\mathsf{pw}^*)', (t^*)'$ such that
$$H_2(\mathsf{pw}^*, T) \oplus t^* = H_2((\mathsf{pw}^*)', T) \oplus (t^*)'$$
(so $\mathcal{A}$ can set s^* to be $s \oplus H_2(\mathsf{pw}^*, T) \oplus t^*$). Note that all four values in the equation are determined by three (different) H_2 queries $H_2(\mathsf{pw}^*, T)$,

[22] The notations are slightly messed up, since the pw^* here has nothing to do with what's called pw^* in Game 13. Essentially, the pw^* here is $\mathcal{A}$'s guess of P''s password (which is called $(\mathsf{pw}')^*$ in the simulator), whereas the pw^* in Game 13 is $\mathcal{A}$'s guess of P's password. The good news is that the pw^* in Game 13 is never reused once Game 13 is done, so there is no potential confusion.

$H_2((\mathsf{pw}^*)', T)$ and $H_2(\mathsf{pw}^*, T^*)$;[23] by an argument similar to above, the probability of this case is upper-bounded by $q_2(q_2-1)(q_2-2)/2^{3n}$.

- There is a collision between type (1) and type (2), i.e., there are $r^*, t^*, (\mathsf{pw}^*)'$, $(t^*)'$ such that
$$r^* \oplus t^* = s \oplus H_2((\mathsf{pw}^*)', T) \oplus (t^*)'$$
(so $\mathcal{A}$ can set s^* to be this value). Note that s is already fixed, and all four other values in the equation are determined by two (different) H_2 queries and one H_1 query; by an argument similar to above, the probability of this case is upper-bounded by $q_1 q_2(q_2-1)/2^{3n}$.

Summing up, we have

$$\mathbf{Dist}_{\mathcal{Z}}^{14,15} \leq \frac{q_2[q_1(q_1+q_2-2)+(q_2-1)(q_2-2)]}{2^{3n}}.$$

This game corresponds to part of game G_2 in [ABJ25] (as aborting conditions [A2a] and [A2b2]). The game in [ABJ25] aborts more often than necessary, though: every time $\mathcal{A}$ makes an $H_2(\star,\star)$ query, the game performs the checks and aborts accordingly, while it is only necessary to go over all of $\mathcal{A}$'s $H_2(\star, T^*)$ queries for the specific T^* as part of $\mathcal{A}$'s message to P'.

Performing the checks on $\mathcal{A}$'s H_2 queries, instead of when $\mathcal{A}$ sends the (s^*, T^*) message, also adds more bookkeeping and complicates the proof. In [ABJ25], after performing the checks, the game (if it does not abort) computes the corresponding KEM public key and stores it together with (pw^*, T^*) (the record is called ForwS); later when $\mathcal{A}$ sends (s^*, T^*), the game first fetches the record to recover pw^*, and if $\mathsf{pw}^* = \mathsf{pw}'$, the game then recovers the KEM public key and uses it to complete the remaining part of P''s algorithm. This creates a conceptual discrepancy with the ideal world, and a game hop (game G_{16} in [ABJ25]) is needed. By performing the checks when $\mathcal{A}$ sends (s^*, T^*), we avoid all these and skip their game G_{16}.

Correct password guess for P'.

Game 16: Again in the case where $(s^*, T^*) \neq (s, T)$, if pw^* is defined in type (2) in Game 15 and $\mathsf{pw}^* = \mathsf{pw}'$, retrieve record $\langle \mathsf{pw}^*, pk^*, \star \rangle$ (there must be one since such a record is created when $\mathcal{A}$ queries $H_2(\mathsf{pw}^*, T)$, as per Game 5) and set $pk' := pk^* \cdot (T^*/T)$.[24]

In Game 15, we have (in P''s algorithm)

$$pk' = \frac{T^*}{R'} = \frac{T^*}{H_1(\mathsf{pw}', r')} = \frac{T^*}{H_1(\mathsf{pw}', s^* \oplus t')} = \frac{T^*}{H_1(\mathsf{pw}', s^* \oplus H_2(\mathsf{pw}', T^*))},$$

[23] Note that in this case $T^* \neq T$ since otherwise $T^* = T \Rightarrow H_2(\mathsf{pw}^*, T) = t^* \Rightarrow s^* = s \oplus H_2(\mathsf{pw}^*, T) \oplus t^* = s$, so $(s^*, T^*) = (s, T)$ which violates the condition of this game.

[24] What's called pk^* here is $(pk')^*$ in the simulator, as there is another pk^* on the P side as defined in Game 13 (cf. Footnote 22).

but also (in the creation of the record $\langle \mathsf{pw}^*, pk^*, \star \rangle$)

$$pk^* = \frac{T}{R^*} = \frac{T}{H_1(\mathsf{pw}^*, r^*)} = \frac{T}{H_1(\mathsf{pw}^*, s \oplus t^*)} = \frac{T}{H_1(\mathsf{pw}^*, s \oplus H_2(\mathsf{pw}^*, T))}.$$

The condition of this game specifies that $\mathsf{pw}^* = \mathsf{pw}'$, and type (2) in Game 15 specifies that $s \oplus H_2(\mathsf{pw}^*, T) = s \oplus H_2(\mathsf{pw}^*, T^*)$. This means that the denominators in the expressions of pk' and pk^* are equal, and thus $pk' = pk^* \cdot (T^*/T)$ — exactly as in Game 15. We have

$$\mathbf{Dist}_{\mathcal{Z}}^{15,16} = 0.$$

This game corresponds to game G_{17} in [ABJ25].

Incorrect password guess for P'.

Game 17: Again in the case where $(s^*, T^*) \neq (s, T)$, if $\mathsf{pw}^* \neq \mathsf{pw}'$ or no pw^* is defined as in Game 15, sample $pk' \leftarrow \mathbb{G}$, compute $(c, \star) \leftarrow \mathsf{Encap}(pk')$, and sample $SK' \leftarrow \{0,1\}^n, \tau' \leftarrow \{0,1\}^{2n}$.

We construct a reduction $\mathcal{R}_{\text{UNI-ANO-RO}}$ to the $(q_1+1)(q_2+1)$-UNI-ANO-RO property of the KEM scheme:

Reduction $\mathcal{R}_{\text{UNI-ANO-RO}}$:

On input $(pk_{1,1}, \ldots, pk_{q_1+1, q_2+1})$,

1. Invoke $\mathcal{Z}$. On $(\mathsf{NewSession}, sid, P, P', \mathsf{pw}, \text{“initiator”})$ from $\mathcal{Z}$, sample $s \leftarrow \{0,1\}^{3n}, T \leftarrow \mathbb{G}$. Send (s, T) to $\mathcal{A}$.

2. Whenever there is an $H_2(\star, T)$ query by $\mathcal{A}$, do what's described in Game 5 (i.e., lines 42–46 in Figure 9). (This in particular yields a record in the form of $\langle \mathsf{pw}^*, pk^*, sk^* \rangle$.) Furthermore, for the j-th query to H_2 (by $\mathcal{A}$ or $\mathcal{R}_{\text{UNI-ANO-RO}}$ itself on behalf of P' — see step 3 below), say it is $H_2(\mathsf{pw}^*, T^*) =: t^*$, and the i-th H_1 query in the form of $H_1(\mathsf{pw}^*, r^*)$, compute
$$s^* := r^* \oplus t^*, \quad R^* := T^*/pk_{i,j}.$$
   (a) If $H_1(\mathsf{pw}^*, r^*)$ has been defined when the $H_2(\mathsf{pw}^*, T^*)$ query happens (either because it was queried *before* the $H_2(\mathsf{pw}^*, T^*) =: t^*$ query or because it has been programmed by $\mathcal{R}_{\text{UNI-ANO-RO}}$ while answering $\mathcal{A}$'s $H_2(\mathsf{pw}^*, T)$ query), store $\langle \mathsf{PasswordGuess}, \mathsf{pw}^*, s^*, T^* \rangle$. // Anchor query, i.e., pw^* is the password guess corresponding to (s^*, T^*) At any time, if there is more than one $\langle \mathsf{PasswordGuess}, \star, s^*, T^* \rangle$ record with the same s^*, T^*, abort.
   (b) Otherwise program $H_1(\mathsf{pw}^*, r^*) := R^*$.

3. On $(\mathsf{NewSession}, sid, P', P, \mathsf{pw}', \text{“respondent”})$ from $\mathcal{Z}$ and (s^*, T^*) from $\mathcal{A}$, if $(s^*, T^*) = (s, T)$, output 1 and halt. If there is a record $\langle \mathsf{PasswordGuess}, \mathsf{pw}', s^*, T^* \rangle$, also output 1 and halt. Otherwise // Incorrect or no password guess (the case that matters in this game hop)

(a) If $\mathcal{A}$ has not queried $H_2(\mathsf{pw}', T^*)$ and/or $H_1(\mathsf{pw}', r')$ (where $r' = s^* \oplus H_2(\mathsf{pw}', T^*)$), make these queries on behalf of P'.

(b) Say $H_2(\mathsf{pw}', T^*)$ is the j^*-th query to H_2 and $H_1(\mathsf{pw}', r')$ is the i^*-th query to H_1 in the form of $H_1(\mathsf{pw}', \star)$. Output (i^*, j^*) to the challenger and receive (c, SK', τ').

(c) Output SK' to $\mathcal{Z}$, and send (c, τ') to $\mathcal{A}$.

4. On (c^*, τ^*) from $\mathcal{A}$,

(a) Retrieve record $\langle \mathsf{pw}, pk, sk \rangle$ stored in step 2. If there is no such record, output $SK := \perp$ to $\mathcal{Z}$.

(b) Find $\mathcal{A}$'s $H_{\mathsf{tag}}(\mathsf{pw}, pk, s, T, c^*, K^*)$ query resulting in τ^*. If there is none or more than one, output $SK := \perp$ to $\mathcal{Z}$.

(c) Compute $K := \mathsf{Decap}(sk, c^*)$. // No need to query a plaintext-checking oracle since pk on the P side and pk' on the P' side are independent of each other, so the reduction can sample sk on its own
If $K = K^*$, output $SK := H_{\mathsf{key}}(K)$ to $\mathcal{Z}$.
If $K \neq K^*$, output $SK := \perp$ to $\mathcal{Z}$.

5. When $\mathcal{Z}$ outputs a bit b^*, copy $\mathcal{Z}$'s output.

We first argue that the pw^* as in record $\langle \mathsf{PasswordGuess}, \mathsf{pw}^*, s^*, T^* \rangle$ is exactly the same as the pw^* extracted from (s^*, T^*) in Game 15:

- If $\mathcal{A}$ queried $H_1(\mathsf{pw}^*, r^*)$ before the $H_2(\mathsf{pw}^*, T^*)$ query, this is exactly type (1) in Game 15.
- If $\mathcal{A}$ queried $H_2(\mathsf{pw}^*, T)$, which triggered the programming of $H_1(\mathsf{pw}^*, r^*)$ before the $H_2(\mathsf{pw}^*, T^*)$ query, the code in Game 5 (i.e., lines 42–46 in Figure 9) specifies that the H_1 instance that is programmed on is $(\mathsf{pw}^*, s \oplus H_2(\mathsf{pw}^*, T))$. Thus,

$$s \oplus H_2(\mathsf{pw}^*, T) = r^* = s^* \oplus H_2(\mathsf{pw}^*, T^*),$$

which is exactly type (2) in Game 15.

Therefore, in step 3 $\mathcal{R}_{\text{UNI-ANO-RO}}$ accurately decides whether the condition in this game, namely "$\mathsf{pw}^* \neq \mathsf{pw}'$ or no pw^* is defined", is satisfied or not.

Next, we argue that $\mathcal{R}_{\text{UNI-ANO-RO}}$ simulates Game 16/17 correctly:

- Suppose $\mathcal{R}_{\text{UNI-ANO-RO}}$'s challenge bit $b = 0$, i.e., $(c, K') \leftarrow \mathsf{Encap}(pk_{i^*,j^*})$ and $SK' = H_{\mathsf{key}}(K'), \tau' = H_{\mathsf{tag}}(\ldots, K')$. For the "special" $H_2(\mathsf{pw}', T^*)$ query and $H_1(\mathsf{pw}', r')$ query, in both Game 16 and $\mathcal{R}_{\text{UNI-ANO-RO}}$'s simulation,

$$s^* = r' \oplus H_2(\mathsf{pw}', T^*), \quad t^* = pk' \cdot H_1(\mathsf{pw}', r');$$

if we sample $H_1(\mathsf{pw}', r') \leftarrow \mathbb{G}$ then we get Game 16, whereas if we sample $pk' \leftarrow \mathbb{G}$ (as pk_{i^*,j^*}) then we get $\mathcal{R}_{\text{UNI-ANO-RO}}$'s simulation. Thus, the distribution of pk' is identical in Game 16 and in $\mathcal{R}_{\text{UNI-ANO-RO}}$'s simulation.

Later, pk' is used to compute c, K' in both Game 16 and $\mathcal{R}_{\text{UNI-ANO-RO}}$'s simulation.
Regarding other $H_2(\mathsf{pw}^*, T^*)$ and $H_1(\mathsf{pw}^*, r^*)$ queries, the $pk_{i,j}$ that $\mathcal{R}_{\text{UNI-ANO-RO}}$ uses while programming $H_1(\mathsf{pw}^*, r^*)$ is uniformly random and independent of the rest of the game, so $H_1(\mathsf{pw}^*, r^*)$ is programmed to a uniformly random value — again the same as Game 16.

- Suppose $\mathcal{R}_{\text{UNI-ANO-RO}}$'s challenge bit $b = 1$, i.e., $(c, \star) \leftarrow \mathsf{Encap}(pk')$ for a freshly sampled pk' and $SK' \leftarrow \{0,1\}^n, \tau' \leftarrow \{0,1\}^{2n}$. *All* $pk_{i,j}$ values that $\mathcal{R}_{\text{UNI-ANO-RO}}$ uses while programming H_1 are uniformly random and independent of the rest of the game, so H_1 is always programmed to a uniformly random value — the same as Game 17. Later, in both Game 17 and $\mathcal{R}_{\text{UNI-ANO-RO}}$'s simulation a freshly sampled pk' is used to compute c, and SK' and τ' are sampled uniformly at random.

We have

$$\mathbf{Dist}_{\mathcal{Z}}^{16,17} \leq \mathbf{Adv}^{(q_1+1)(q_2+1)\text{-UNI-ANO-RO}}_{\mathcal{R}_{\text{UNI-ANO-RO}}}.$$

This game corresponds to games G_{18} and G_{19} in [ABJ25]. [ABJ25] has two games: G_{18} that changes SK' and τ' (via a reduction to the OW-PCA property of the KEM scheme), and G_{19} that changes c (via a reduction to the ANO-PCA property of the KEM scheme). While we believe their argument is fundamentally correct, it could be significantly improved:

1. *[ABJ25] reduces to OW-PCA and ANO-PCA,[25] which is unnecessary: OW and ANO suffice.*
2. *Their simulation strategy is slightly different from ours: they use pk' generated by Gen, rather than sampled uniformly at random. Note that in this "incorrect password guess" case, in the real world $pk' = T^*/H_1(\mathsf{pw}', r')$ is already uniformly random, so this is fundamentally a KEM property about encapsulation of uniformly random (rather than honestly generated) public keys and we believe it is clearer to present it in this way.*[26]
3. *Their honest generation of pk' also means that their reductions cannot simulate the games perfectly, since they program $H_1(\mathsf{pw}', r')$ using pk' which is not uniformly random — while it "should" be uniformly random. As such, they have do a reduction to UNI-PK within the OW-PCA/ANO-PCA reduction.*
4. *What complicates things even further is that none of their (or our) reductions knows which of the $H_1(\mathsf{pw}', r')$ queries should be programmed using*

[25] [ABJ25] actually mentions ANO-CPA, which appears to be a typo.

[26] By contrast, in the $(s^*, T^*) = (s, T)$ case pk' has to be generated honestly, which is our Games 3, 4, 6 and 7: If $\mathsf{pw} = \mathsf{pw}' \wedge (s^*, T^*) = (s, T)$ then sk corresponding to $pk = pk'$ is used on the P side (in decapsulation), so pk' cannot be changed to uniformly random; and the $\mathsf{pw} \neq \mathsf{pw}' \wedge (s^*, T^*) = (s, T)$ case cannot be separated from the $\mathsf{pw} = \mathsf{pw}' \wedge (s^*, T^*) = (s, T)$ case (since the simulator never knows if $\mathsf{pw} = \mathsf{pw}'$ or not), so pk' cannot be changed to uniformly random in this case either.

pk', since $\mathcal{A}$ can make many 2F decryption efforts, resulting in many H_1 values to be programmed, before *sending (s^*, T^*) to P' (cf. the discussion after Definition 6). This means that their UNI-PK reduction cannot be moved "out of" the OW-PCA/ANO-PCA reduction by defining a separate game that uses honestly generated pk' to program $H_1(\mathsf{pw}', r')$, as that would require the* game *to make a guess! This "reduction within a reduction" would better be filtered out as a separate helper security property.*

5. *Finally, the reductions in [ABJ25] appear to never check for the "related key attack", i.e., from their proof it is unclear what our Game 16 (or their game G_{17}) is for.*

We believe it is much cleaner to define a tailored KEM property that allows a single *reduction to go through, which is UNI-ANO-RO. In this way, we can separate out the multiple reductions to basic KEM properties, as well as the hybrid argument, on the level of KEM; this makes our OEKE security argument easier to understand.*[27]

Summary. We summarize the changes since Game 11 (i.e., after Figure 9):

1. *P''s session key and P'-to-P message:* On (s^*, T^*) from $\mathcal{A}$ to P', if $(s^*, T^*) \neq (s, T)$, Game 17 extracts $(\mathsf{pw}')^*$ from (s^*, T^*) using the method in Game 15. Then
   (a) If $(\mathsf{pw}')^* \neq \mathsf{pw}'$, then Game 17 outright samples SK', τ' at random. Regarding c, Game 17 samples $pk' \leftarrow \mathbb{G}$ and computes $(c, \star) \leftarrow \mathsf{Encap}(pk')$. [changed in Game 17]
   (b) If $(\mathsf{pw}')^* = \mathsf{pw}'$, Game 15 specifies two types of extraction of $(\mathsf{pw}')^*$. There is no change from the real world for type (1). For type (2), Game 17 computes $pk' := (pk')^* \cdot (T^*/T)$ (and uses pk' to compute c', SK', τ'). [changed in Game 16]
2. *P's session key:* On (c^*, τ^*) from $\mathcal{A}$ to P, we have made the following changes:
   (a) If $\mathsf{pw} \neq \mathsf{pw}' \wedge (s^*, T^*) = (s, T) \wedge (c^*, \tau^*) = (c, \tau')$, Game 17 outright sets $SK := \perp$. [changed in Game 14]
   (b) If $\neg((s^*, T^*) = (s, T) \wedge (c^*, \tau^*) = (c, \tau'))$ (i.e., $\mathcal{A}$ is not passive), Game 17 retrieves two records:
      - $\mathcal{A}$'s $H_{\mathsf{tag}}(\mathsf{pw}^*, pk^*, s, T, c^*, K^*)$ query resulting in τ^*. If there is none or more than one, or if there is a unique such query with $\mathsf{pw}^* \neq \mathsf{pw}$, Game 17 sets $SK := \perp$.

[27] In principle, we could also combine Games 3 and 4, as well as Games 6 and 7, via a single reduction to some "integrated" KEM properties. We choose not to do it as we wish to follow the proof structure in [ABJ25] as much as possible. In contrast, [JRX25, Definition 3.6] has a combined KEM property called "pseudorandom non-malleability".

- $\langle \mathsf{pw}^*, pk^*, sk^* \rangle$ record stored when $\mathcal{A}$ queries $H_2(\mathsf{pw}^*, T)$. If there is none, Game 17 sets $SK := \perp$.

(Note that pw^* and pk^* in these two records have to match each other.) After that, Game 17 checks if $K^* = \mathsf{Decap}(sk^*, c^*)$, and computes SK using K^* or sets $SK := \perp$ accordingly. [changed in Games 11, 12, 13]

See Figure 10 for a summary of Game 17.

Comparison with the ideal world. We now claim that Game 17 is identical to the ideal world, except that the challenger in Game 17 is split into the $\mathcal{F}_{\text{PAKE-1EA}}$ functionality and the simulator $\mathcal{S}$ in the ideal world (which, of course, does not make a difference in $\mathcal{Z}$'s view).

item to be simulated	case	Game 17 (Figure 10)	simulator (Figures 6 and 7)
P-to-P' message (s, T)		line 2	the only step
P''s session key SK' and P'-to-P message (c, τ')	$(s^*, T^*) = (s, T)$	lines 4–7	step 1
	no $(\mathsf{pw}')^*$, or $(\mathsf{pw}')^* \neq \mathsf{pw}'$	lines 10–13	step 2(b)(i)
	$(\mathsf{pw}')^* = \mathsf{pw}'$	lines 14–25	step 2(b)(ii)
P's session key SK	$(s^*, T^*) = (s, T) \wedge (c^*, \tau^*) = (c, \tau')$	lines 27–30	step 1
	no pw^*, or more than one pw^*	lines 33–34	step 2(a)
	no $\langle \mathsf{pw}^*, pk^*, sk^* \rangle$ record	lines 38–39	step 2(b)
	$K^* \neq K$	lines 43–44	step 2(c)
	$\mathsf{pw}^* \neq \mathsf{pw}$	lines 35–36	step 2(d)(i)
	$\mathsf{pw}^* = \mathsf{pw} \wedge K^* = K$	lines 41–42	step 2(d)(ii)
$\mathcal{A}$'s $H_2(\star, T)$ queries		lines 46–51	the only step

Table 5: Comparison between Game 17 and the ideal world. The "step" in simulator refers to the step in Figures 6 and 7 under the corresponding title. Various aborting conditions are omitted

From Table 5 (and after going through some simple but tedious checks), we can see that Game 17 and the ideal world are exactly identical. This completes the proof.

5 Security Analyses of Other Versions (I): Including (s, T) in the Hash

We assume the reader is familiar with the security proof in Section 4, as the simulator and many games there will be frequently referred to or reused.

5.1 What Will Go Wrong If You Don't Hash Everything?

One might fear that every new variant of OEKE — with some inputs removed from H_{tag} — needs a new security analysis from scratch, causing this paper

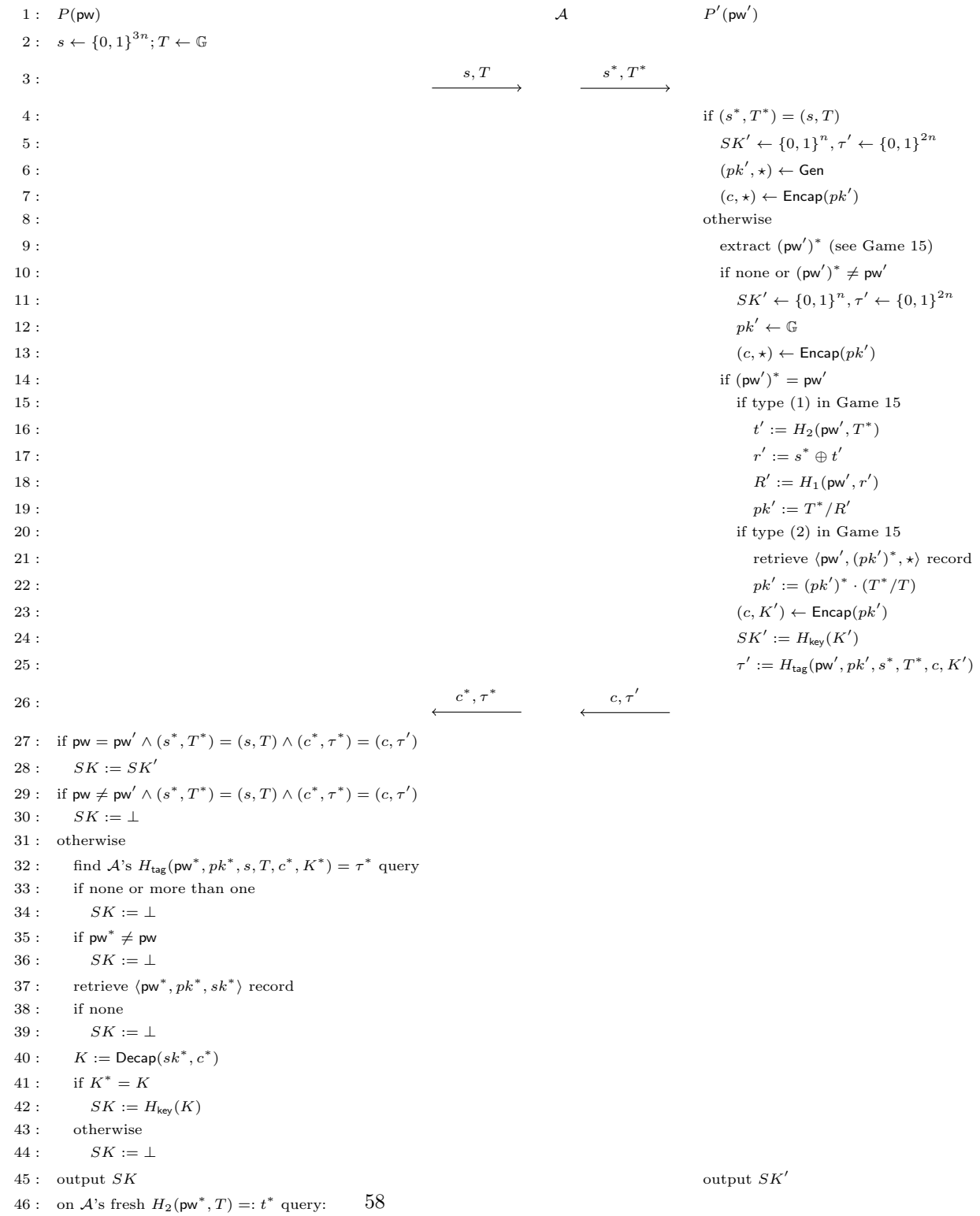

Figure 10: Game 17 (with various aborting conditions omitted)

to be (length of Section 4) $\times 16 \approx 430$ pages long. Fortunately, most of the simulator and the game hops from the proof in Section 4 can be reused. This, in fact, should be expected: only the definition of the P'-to-P authenticator τ' changes in these variants, so the only parts of the proof that are affected are P's behavior — in particular, how to extract the password guess and when to output $\perp$ — on the P'-to-P message.

Concretely, Games 2, 11 and 13 in Section 4.2 need stronger KEM security properties, or at least a new analysis, to go through. We explained in *underlined* comments in each of them why the current arguments critically require pw, pk, (s, T) and/or c as part of the H_{tag} inputs, which we summarize in Table 6:

#	summary	what needs to be included in H_{tag}
2	extract K^* from τ^*	$\mathsf{pw}, (s, T), c$
11	reject if $\mathcal{A}$ does not make an $H_2(\mathsf{pw}, T)$ query	pk
13	extract pw^* from τ^* and check against K^*	pw

Table 6: Summary of games in Section 4.2 that require pw, pk and/or c as part of H_{tag} inputs

In this section, we analyze seven variants: those with (s, T) included in H_{tag}, but at least one of pw, pk, c removed. In Appendix B we analyze the variants where (s, T) are removed from H_{tag}.

5.2 The "Hash pw, pk and (s, T)" Version

We start from the version where τ is defined as $H_{\mathsf{tag}}(\mathsf{pw}, pk, s, T, K)$, i.e., c is removed from the hash. Table 6 suggests that only Game 2 needs to change, and the simulator and all other games can be reused in the proof.

Looking more closely, only Claim 1 in Game 2 needs a new proof: the argument after this claim also does not change if we remove c from H_{tag}. Below we state and prove the new claim.

Claim 2. *Suppose the CBIND property holds for the KEM scheme. In the case where $\neg(\mathsf{pw} = \mathsf{pw}' \wedge (s^*, T^*) = (s, T) \wedge c^* = c)$, the probability that P' queries $H_{\mathsf{tag}}(\mathsf{pw}, pk, s, T, K)$ is negligible.*

Proof. P' makes a single H_{tag} query, on $(\mathsf{pw}', pk', s^*, T^*, K')$. If $\mathsf{pw} \neq \mathsf{pw}' \vee (s^*, T^*) \neq (s, T)$, then of course P''s query is not $H_{\mathsf{tag}}(\mathsf{pw}, pk, s, T, K)$. Now assume $\mathsf{pw} = \mathsf{pw}' \wedge (s^*, T^*) = (s, T)$. This implies $pk = pk'$ by the definitions of pk and pk', and $c^* \neq c$ by the condition of the claim.

We now upper-bound $\Pr[K = K']$. Recall that K is defined as $\mathsf{Decap}(sk, c^*)$, so $\Pr[K = K'] = \Pr[\mathsf{Decap}(sk, c^*) = K']$. We upper-bound this probability via a reduction $\mathcal{R}_{\text{CBIND}}$ to the CBIND property of the KEM scheme:

Reduction $\mathcal{R}_{\text{CBIND}}$:

On input (pk, c, K'),

1. Invoke $\mathcal{Z}$. On $(\mathsf{NewSession}, sid, P, P', \mathsf{pw}, \text{“initiator”})$ from $\mathcal{Z}$, run P's algorithm to compute (s, T), except that $\mathcal{R}_{\text{CBIND}}$ uses its input pk instead of $(pk, \star) \leftarrow \mathsf{Gen}$. Send (s, T) to $\mathcal{A}$.

2. On $(\mathsf{NewSession}, sid, P', P, \mathsf{pw}', \text{“respondent”})$ from $\mathcal{Z}$ and (s^*, T^*) from $\mathcal{A}$, if $\neg(\mathsf{pw} = \mathsf{pw}' \wedge (s^*, T^*) = (s, T))$, abort. Otherwise compute $SK' := H_{\mathsf{key}}(K')$ and $\tau' := H_{\mathsf{tag}}(\mathsf{pw}, pk, s, T, c, K')$. Output SK' to $\mathcal{Z}$, and send (c, τ') to $\mathcal{A}$.

3. On (c^*, τ^*) from $\mathcal{A}$, if $c^* = c$, abort. Otherwise output c^*.

If $\mathcal{A}$ sends c^* such that $\mathsf{Decap}(sk, c^*) = K'$, then $\mathcal{R}_{\text{CBIND}}$ wins the CBIND game. Thus,

$$\Pr[\mathsf{Decap}(sk, c^*) = K'] \leq \mathbf{Adv}^{\text{CBIND}}_{\mathcal{R}_{\text{CBIND}}}.$$

If $\mathsf{Decap}(sk, c) \neq K'$, i.e., $K \neq K'$, then of course P''s query is not $H_{\mathsf{tag}}(\mathsf{pw}, pk, s, T, K)$. This yields the claim. □

We conclude that the “hash pw, pk and (s, T)” version is secure under the CBIND property of the KEM scheme (in addition to UNI-PK, OW-1PCA and ANO-1PCA that are already needed in the “hash everything” version).

5.3 The “Hash pw and (s, T)” Version

We now turn to the variant where pk is further removed from H_{tag} inputs, i.e., τ is defined as $H_{\mathsf{tag}}(\mathsf{pw}, s, T, K)$. Table 6 suggests that the argument under Game 11 in Section 4.2 need to (essentially) change. (Of course, since c is also removed from H_{tag}, the change in Section 5.2 also needs to be made.) Below we repeat Game 11 and make the new argument.

Game 11: In the case where $\neg(\mathsf{pw} = \mathsf{pw}' \wedge (s^*, T^*) = (s, T) \wedge c^* = c)$, if there is no $\langle \mathsf{pw}, pk, sk \rangle$ record, set $SK := \perp$ (i.e., replace the brown text in Figure 9 with “$SK := \perp$”).

Game 10 and Game 11 differ only if $\mathcal{A}$ does not make an $H_2(\mathsf{pw}, T)$ query, but makes an $H_{\mathsf{tag}}(\mathsf{pw}, s, T, K^*)$ query resulting in τ^*, where $K^* = K = \mathsf{Decap}(sk, c^*)$ for a freshly generated sk. We upper-bound this probability via a reduction $\mathcal{R}_{\text{UNPRE}}$ to the UNPRE property of the KEM scheme:

Reduction $\mathcal{R}_{\text{UNPRE}}$:

1. Invoke $\mathcal{Z}$. On $(\mathsf{NewSession}, sid, P, P', \mathsf{pw}, \text{“initiator”})$ from $\mathcal{Z}$, sample $s \leftarrow \{0,1\}^{3n}, T \leftarrow \mathbb{G}$. Send (s, T) to $\mathcal{A}$.

2. On $(\mathsf{NewSession}, sid, P', P, \mathsf{pw}', \text{“respondent”})$ from $\mathcal{Z}$ and (s^*, T^*) from $\mathcal{A}$, run P''s algorithm in Game 10 to compute c, SK' and τ'. Output SK' to $\mathcal{Z}$, and send (c, τ') to $\mathcal{A}$.

3. On (c^*, τ^*) from $\mathcal{A}$, if $\mathsf{pw} = \mathsf{pw}' \wedge (s^*, T^*) = (s, T) \wedge c^* = c$, abort. Otherwise find $\mathcal{A}$'s $H_{\mathsf{tag}}(\mathsf{pw}, s, T, K^*)$ query resulting in τ^*. If there is none or more than one, abort. Otherwise output (c^*, K^*).

4. On $\mathcal{A}$'s $H_2(\mathsf{pw}^*, T)$ query, if $\mathsf{pw}^* = \mathsf{pw}$, abort. Otherwise do what Game 10 specifies.

It is straightforward to see that if the "bad event" above happens, then $\mathcal{R}_{\text{UNPRE}}$ wins its UNPRE game. We have

$$\mathbf{Dist}_{\mathcal{Z}}^{10,11} \leq \mathbf{Adv}_{\mathcal{R}_{\text{UNPRE}}}^{\text{UNPRE}}.$$

Overall, we conclude that the "hash pw and (s, T)" version is secure under the CBIND and UNPRE properties of the KEM scheme.

The "hash pw, (s, T) and c" version. Note that the change in Game 2 (which requires CBIND) and the change in Game 11 (which require UNPRE) are unrelated to each other. Therefore, if we define τ as $H_{\mathsf{tag}}(\mathsf{pw}, s, T, c, K)$, then only Game 11 needs to change, and the protocol is secure under the UNPRE property of the KEM scheme.

5.4 The "Hash pk and (s, T)" Version

We now consider the variant where pw, but not pk, is further removed from H_{tag} inputs, i.e., τ is defined as $H_{\mathsf{tag}}(pk, s, T, K)$. Table 6 suggests that Games 2 and 13 in Section 4.2 need to (essentially) change. In fact the simulator also needs to change a bit, which we start from.

5.4.1 Changes to the Simulator

The only changes to the simulator $\mathcal{S}$ appear when $\mathcal{A}$ sends (c^*, τ^*) to P and $\mathcal{S}$ needs to extract a password guess pw^* from it. In the "hash everything" version (Section 4.1), τ^* is computed as $H_{\mathsf{tag}}(\mathsf{pw}^*, pk^*, s, T, c^*, K^*)$, so $\mathcal{S}$ can extract pw^* directly from $\mathcal{A}$'s H_{tag} queries and *check its consistency with* pk^*, c^*, K^* (see the last paragraph of Section 4.1.1). Here, $\tau^* = H_{\mathsf{tag}}(pk^*, s, T, K^*)$, which prevents $\mathcal{S}$ from reading off pw^* from $\mathcal{A}$'s H_{tag} queries. However, the inclusion of pk^* still allows $\mathcal{S}$ to find the corresponding $\langle \mathsf{pw}^*, pk^*, sk^* \rangle$ record, from which $\mathcal{S}$ can recover pw^* and check for consistency. Still, there is a negligible chance that there are multiple $\langle \mathsf{pw}^*, pk^*, sk^* \rangle$ records for the same pk^*, which we must rule out.

We show the new simulator in Figure 11. (Only the steps after receiving (c^*, τ^*) are included.)

P's session key

On (c^*, τ^*) from $\mathcal{F}$, simulate P's output as follows:

1. If $(s^*, T^*) = (s, T) \wedge (c^*, \tau^*) = (c, \tau')$, send $(\mathsf{NewKey}, sid, P, 0^n)$ to $\mathcal{F}$. // $\mathcal{A}$ is passive
2. Otherwise find $\mathcal{A}$'s $H_{\mathsf{tag}}(pk^*, s, T, K^*)$ query resulting in τ^*.
   (a) If there is no such query or more than one such query, send $(\mathsf{TestPwd}, sid, P, \perp)$ and then $(\mathsf{NewKey}, sid, P, 0^n)$ to $\mathcal{F}$.
   (b) If there is exactly one such query, retrieve record $\langle \mathsf{pw}^*, pk^*, sk^* \rangle$. If there is more than one such record, abort. If there is no such record, send $(\mathsf{TestPwd}, sid, P, \perp)$ and then $(\mathsf{NewKey}, sid, P, 0^n)$ to $\mathcal{F}$.
   (c) Compute $K := \mathsf{Decap}(sk^*, c^*)$. If $K^* \neq K$, send $(\mathsf{TestPwd}, sid, P', \perp)$ and then $(\mathsf{NewKey}, sid, P, 0^n)$ to $\mathcal{F}$.
   (d) Send $(\mathsf{TestPwd}, sid, P, \mathsf{pw}^*)$ to $\mathcal{F}$.
      i. If $\mathcal{F}$ replies with "wrong guess", send $(\mathsf{NewKey}, sid, P, 0^n)$ to $\mathcal{F}$.
      ii. If $\mathcal{F}$ replies with "correct guess", compute $SK := H_{\mathsf{key}}(K)$ and send $(\mathsf{NewKey}, sid, P, SK)$ to $\mathcal{F}$.

Figure 11: UC simulator $\mathcal{S}$ for the "hash pk and (s, T)" variant of OEKE. Only the paragraph that needs essential change (shown in blue) is included

5.4.2 Changes in Game Hops

Change in Game 2. The following claim replaces Claim 1 in the "hash everything" version:

Claim 3. *Suppose the CBIND property holds for the KEM scheme. In the case where* $\neg(\mathsf{pw} = \mathsf{pw}' \wedge (s^*, T^*) = (s, T) \wedge c^* = c)$*, the probability that* P' *queries* $H_{\mathsf{tag}}(pk, s, T, K)$ *is negligible.*

Proof. P' makes a single H_{tag} query, on (pk', s^*, T^*, K'). If $(s^*, T^*) \neq (s, T)$, then of course P''s query is not $H_{\mathsf{tag}}(pk, s, T, K)$. Now assume $(s^*, T^*) = (s, T)$, which implies $\mathsf{pw} \neq \mathsf{pw}' \vee c^* \neq c$ by the condition of the claim.

- If $\mathsf{pw} \neq \mathsf{pw}'$, we have

$$pk = \frac{T}{R} = \frac{T}{H_1(\mathsf{pw}, r)} = \frac{T}{H_1(\mathsf{pw}, s \oplus t)},$$
$$pk' = \frac{T}{R'} = \frac{T}{H_1(\mathsf{pw}', r')} = \frac{T}{H_1(\mathsf{pw}', s \oplus t')}.$$

  Since $\mathsf{pw} \neq \mathsf{pw}'$, $H_1(\mathsf{pw}, s \oplus t)$ and $H_1(\mathsf{pw}', s \oplus t')$ are independent of each other, so $\Pr[pk = pk'] = \Pr[H_1(\mathsf{pw}, s \oplus t) = H_1(\mathsf{pw}', s \oplus t')] = 1/|\mathbb{G}|$.

- If $\mathsf{pw} = \mathsf{pw}' \wedge c^* \neq c$, then $\mathsf{pw} = \mathsf{pw}' \wedge (s^*, T^*) = (s, T)$, which implies that $pk = pk'$ by the definition of pk and pk'. Then we can show that $\Pr[K = K']$ is negligible, via a reduction $\mathbf{Adv}^{\mathrm{CBIND}}_{\mathcal{R}_{\mathrm{CBIND}}}$ similar to that in the proof of Claim 2. We have $\Pr[K = K'] \leq \mathbf{Adv}^{\mathrm{CBIND}}_{\mathcal{R}_{\mathrm{CBIND}}}$.

If $pk \neq pk'$ or $K \neq K'$ then of course P''s query is not $H_{\mathsf{tag}}(pk, s, T, K)$. Overall, we have

$$\Pr[P' \text{ queries } H_{\mathsf{tag}}(pk, s, T, K)] \leq \mathbf{Adv}^{\mathrm{CBIND}}_{\mathcal{R}_{\mathrm{CBIND}}} + \frac{1}{|\mathbb{G}|},$$

which yields the claim. □

The remaining argument is essentially identical to that in the old Game 2.

Change in Game 13. We first state the new Game 13, and then make the argument.

Game 13: In the case where $\neg(\mathsf{pw} = \mathsf{pw}' \wedge (s^*, T^*) = (s, T) \wedge (c^*, \tau^*) = (c, \tau'))$, find $\mathcal{A}$'s $H_{\mathsf{tag}}(pk^*, s, T, K^*)$ query resulting in τ^*. If there is none or more than one, set $SK := \perp$. Otherwise there is a unique such $H_{\mathsf{tag}}(pk^*, s, T, K^*)$ query, and

- If there is more than one $\langle \mathsf{pw}^*, pk^*, sk^* \rangle$ record, abort.
- If there is no $\langle \mathsf{pw}^*, pk^*, sk^* \rangle$ record, set $SK := \perp$.
- If there is a unique $\langle \mathsf{pw}^*, pk^*, sk^* \rangle$ record, but $\mathsf{pw}^* \neq \mathsf{pw}$, also set $SK := \perp$.
- If there is a unique $\langle \mathsf{pw}^*, pk^*, sk^* \rangle$ record, and $\mathsf{pw}^* = \mathsf{pw}$, compute $K := \mathsf{Decap}(sk^*, c^*)$. If $K^* = K$, set $SK := H_{\mathsf{key}}(K)$; otherwise set $SK := \perp$.

If $\mathcal{A}$ makes no $H_{\mathsf{tag}}(\star, s, T, \star)$ query resulting in τ^*, then in particular $\mathcal{A}$ makes no $H_{\mathsf{tag}}(pk, s, T, \star)$ query resulting in τ^*, so both Game 12 and Game 13 set $SK := \perp$. The probability that $\mathcal{A}$ makes two different $H_{\mathsf{tag}}(\star, s, T, \star)$ queries both resulting in τ^* is upper-bounded by $q_{\mathsf{tag}}(q_{\mathsf{tag}} - 1)/2^{2n+1}$. (This part does not change from the argument under the old Game 13.) Below we assume that $\mathcal{A}$ makes a unique $H_{\mathsf{tag}}(pk^*, s, T, K^*)$ query resulting in τ^*:

- We first upper-bound the probability that there is more than one $\langle \mathsf{pw}^*, pk^*, sk^* \rangle$ record for the same pk^*. Note that in each such record, $(pk^*, \star) \leftarrow \mathsf{Gen}$ is freshly generated, so the probability that there is a collision in pk^* is upper-bounded by $\binom{q_2}{2} \cdot p_{\mathrm{guess}}$ (recall that there is a $\langle \mathsf{pw}^*, pk^*, sk^* \rangle$ record for each $H_2(\mathsf{pw}^*, T)$ query).
- If there is no $\langle \mathsf{pw}^*, pk^*, sk^* \rangle$ record, then both Game 12 and Game 13 set $SK := \perp$.

- Next, suppose there is a unique $\langle \mathsf{pw}^*, pk^*, sk^* \rangle$ record, but $\mathsf{pw}^* \neq \mathsf{pw}$. If there is no $\langle \mathsf{pw}, pk, sk \rangle$ record, then both Game 12 and Game 13 set $SK := \perp$. If there is a $\langle \mathsf{pw}, pk, sk \rangle$ record, then $(pk, \star) \leftarrow \mathsf{Gen}$ is freshly generated, so $\Pr[pk^* = pk] \leq q_2 \cdot p_{\text{guess}}$; and if $pk^* \neq pk$, then $\mathcal{A}$ does not make an $H_{\mathsf{tag}}(\mathsf{pw}, s, T, pk)$ query resulting in τ^* (because there is only one H_{tag} query resulting in τ^*, and it is on pk^*), so again both Game 12 and Game 13 set $SK := \perp$.
- Finally, if there is a unique $\langle \mathsf{pw}^*, pk^*, sk^* \rangle$ record, and $\mathsf{pw}^* = \mathsf{pw}$, then Game 13 just runs P's algorithm in Game 12, so these two games are identical.

In sum, we have

$$\mathbf{Dist}_{\mathcal{Z}}^{12,13} \leq \frac{q_2(q_2+1)}{2} \cdot p_{\text{guess}} + \frac{q_{\mathsf{tag}}(q_{\mathsf{tag}}-1)}{2^{2n+1}}.$$

Overall, we conclude that the "hash pk and (s,T)" version is secure under the CBIND property of the KEM scheme.

The "hash pk, (s,T) and c" version. If we define τ as $H_{\mathsf{tag}}(pk, s, T, c, K)$, then the second bullet of the proof of Claim 3 is not needed, and the protocol can be proven secure without the CBIND property of the KEM scheme (i.e., under the same assumptions as in the "hash everything" version).

5.5 The "Only Hash (s,T)" Version

Finally, we consider the variant where both pw and pk (as well as c) are removed from H_{tag} inputs, i.e., τ is defined as $H_{\mathsf{tag}}(s, T, K)$. This is *not* merely a combination of what we did while removing pw and while removing pk (Sections 5.3 and 5.4), and requires a new analysis. Again we will start from the changes to the simulator.

5.5.1 Changes to the Simulator

When pw is removed from H_{tag} inputs in Section 5.4, the simulator needs to change how it extracts $\mathcal{A}$'s password guess pw^*: it cannot read off pw^* from $\mathcal{A}$'s H_{tag} queries, and instead needs to read off pk^* and then fetch the $\langle \mathsf{pw}^*, pk^*, sk^* \rangle$ record. Here, neither pw^* nor pk^* can be read off from $\mathcal{A}$'s H_{tag} queries, and a new extraction strategy is needed. $\mathcal{S}$ can only read off K^*, so it has to go over *all* $\langle \mathsf{pw}^*, pk^*, sk^* \rangle$ records and check which one is consistent with K^* and c^*. The new simulator is shown in Figure 12.

P's session key

On (c^*, τ^*) from $\mathcal{F}$, simulate P's output as follows:

1. If $(s^*, T^*) = (s, T) \wedge (c^*, \tau^*) = (c, \tau')$, send $(\mathsf{NewKey}, sid, P, 0^n)$ to $\mathcal{F}$. // $\mathcal{A}$ is passive
2. Otherwise find $\mathcal{A}$'s $H_{\mathsf{tag}}(s, T, K^*)$ query resulting in τ^*.
   (a) If there is no such query or more than one such query, send $(\mathsf{TestPwd}, sid, P, \perp)$ and then $(\mathsf{NewKey}, sid, P, 0^n)$ to $\mathcal{F}$.
   (b) If there is exactly one such query, retrieve all records in the form of $\langle \mathsf{pw}^*, \star, sk^* \rangle$. For each such record, compute $K := \mathsf{Decap}(sk^*, c^*)$ and check if $K^* = K$. If there is more than one such record, abort. If there is no such record, send $(\mathsf{TestPwd}, sid, P, \perp)$ and then $(\mathsf{NewKey}, sid, P, 0^n)$ to $\mathcal{F}$.
   (c) If there is exactly one record $\langle \mathsf{pw}^*, \star, sk^* \rangle$ with $K := \mathsf{Decap}(sk^*, c^*) = K^*$, send $(\mathsf{TestPwd}, sid, P, \mathsf{pw}^*)$ to $\mathcal{F}$.
      i. If $\mathcal{F}$ replies with "wrong guess", send $(\mathsf{NewKey}, sid, P, 0^n)$ to $\mathcal{F}$.
      ii. If $\mathcal{F}$ replies with "correct guess", compute $SK := H_{\mathsf{key}}(K)$ and send $(\mathsf{NewKey}, sid, P, SK)$ to $\mathcal{F}$.

Figure 12: UC simulator $\mathcal{S}$ for the "only hash (s, T)" variant of OEKE. Only the paragraph that needs essential change (shown in blue) is included

5.5.2 Changes in Game Hops

Change in Game 2. The following claim replaces Claim 1 in the "hash everything" version:

Claim 4. *Suppose the CBIND and KBIND properties hold for the KEM scheme. In the case where* $\neg(\mathsf{pw} = \mathsf{pw}' \wedge (s^*, T^*) = (s, T) \wedge c^* = c)$*, the probability that* P' *queries* $H_{\mathsf{tag}}(s, T, K)$ *is negligible.*

Proof. P' makes a single H_{tag} query, on (s^*, T^*, K'). If $(s^*, T^*) \neq (s, T)$, then of course P''s query is not $H_{\mathsf{tag}}(s, T, K)$. Now assume $(s^*, T^*) = (s, T)$, which implies $\mathsf{pw} \neq \mathsf{pw}' \vee c^* \neq c$ by the condition of the claim.

- If $\mathsf{pw} \neq \mathsf{pw}'$, by the proof of Claim 3, pk and pk' are independent of each other (where pk is honestly generated and pk' is uniformly random). This suggests that we should upper-bound $\Pr[K = K'] = \Pr[\mathsf{Decap}(sk, c^*) = K']$ via a reduction $\mathcal{R}_{\mathrm{KBIND}}$ to the KBIND property of the KEM scheme:

  Reduction $\mathcal{R}_{\mathrm{KBIND}}$:

  Sample $j \leftarrow [q_2 + 1]$ as a guess that the j-th H_2 query (by either $\mathcal{A}$ or P') is $H_2(\mathsf{pw}', T)$.
  On input (pk, pk', c, K'),

1. Invoke $\mathcal{Z}$. On $(\mathsf{NewSession}, sid, P, P', \mathsf{pw}, \text{“initiator”})$ from $\mathcal{Z}$, run P's algorithm to compute (s, T), except that $\mathcal{R}_{\text{KBIND}}$ uses its input pk instead of $(pk, \star) \leftarrow \mathsf{Gen}$. Send (s, T) to $\mathcal{A}$.
2. On the j-th $H_2(\mathsf{pw}^*, T) =: t'$ query (by either $\mathcal{A}$ or $\mathcal{R}_{\text{KBIND}}$ itself on behalf of P' — see step 3 below), compute

$$r' := s \oplus t', \quad R' := T/pk'.$$

(pk' is part of $\mathcal{R}_{\text{KBIND}}$'s input.) If $H_1(\mathsf{pw}^*, r')$ has already been defined and it is not R', abort. Otherwise program $H_1(\mathsf{pw}^*, r') := R'$. // Similar to how H_1 is programmed in Section 4.2, Game 5
3. On $(\mathsf{NewSession}, sid, P', P, \mathsf{pw}', \text{“respondent”})$ from $\mathcal{Z}$ and (s^*, T^*) from $\mathcal{A}$, if $\neg(\mathsf{pw} \neq \mathsf{pw}' \wedge (s^*, T^*) = (s, T))$, abort. Otherwise
   (a) If $\mathcal{A}$ has not queried $H_2(\mathsf{pw}', T)$, make this query on behalf of P'.
   (b) Compute $SK' := H_{\mathsf{key}}(K')$ and $\tau' := H_{\mathsf{tag}}(s, T, K')$. Output SK' to $\mathcal{Z}$, and send (c, τ') to $\mathcal{A}$. (c and K' are part of $\mathcal{R}_{\text{KBIND}}$'s input.)
4. On (c^*, τ^*) from $\mathcal{A}$, output c^*.

By an argument similar to that under Game 5 in Section 4.2, $\mathcal{R}_{\text{KBIND}}$'s aborting probability in step 2 is upper-bounded by $q_1(q_2+1)/2^{3n}$. Assuming $\mathcal{R}_{\text{KBIND}}$ does not abort in step 2, $\mathsf{Decap}(sk, c^*) = K'$, and $\mathcal{R}_{\text{KBIND}}$'s guess of index j is correct, the only difference between Game 1/2 and $\mathcal{R}_{\text{KBIND}}$'s simulation (until $\mathcal{A}$ sends c^*) is that $\mathcal{R}_{\text{KBIND}}$ uses its input pk' to program $H_1(\mathsf{pw}', r')$, and then uses its inputs $(c, K') \leftarrow \mathsf{Encap}(pk')$ as the P'-to-P message and compute SK' and τ'. Since (c, K') are also computed as $\mathsf{Encap}(pk')$ in Game 1/2, $\mathcal{R}_{\text{KBIND}}$ simulates Game 1/2 perfectly. Furthermore, $\mathcal{R}_{\text{KBIND}}$ wins the KBIND game. We have

$$\Pr[K = K' \mid \mathsf{pw} \neq \mathsf{pw}'] \leq (q_2 + 1)\left(\mathbf{Adv}^{\text{KBIND}}_{\mathcal{R}_{\text{KBIND}}} + \frac{q_1}{2^{3n}}\right).$$

- If $\mathsf{pw} = \mathsf{pw}' \wedge c^* \neq c$, then the argument in the proof of Claim 2 still holds, and $\Pr[K = K' \mid \mathsf{pw} = \mathsf{pw}' \wedge c^* \neq c] \leq (q_2 + 1)\mathbf{Adv}^{\text{CBIND}}_{\mathcal{R}_{\text{CBIND}}}$.

If $K \neq K'$ then of course P''s query is not $H_{\mathsf{tag}}(s, T, K)$. Overall, we have

$$\Pr[P' \text{ queries } H_{\mathsf{tag}}(s, T, K)] = \Pr[K = K'] \leq (q_2+1)\left(\mathbf{Adv}^{\text{CBIND}}_{\mathcal{R}_{\text{CBIND}}} + \mathbf{Adv}^{\text{KBIND}}_{\mathcal{R}_{\text{KBIND}}} + \frac{q_1}{2^{3n}}\right),$$

which yields the claim. □

The remaining argument is essentially identical to that in the old Game 2.

Change in Game 11. Game 10 and Game 11 differ only if $\mathcal{A}$ does not make an $H_2(\mathsf{pw}, T)$ query, but makes an $H_{\mathsf{tag}}(s, T, K^*)$ query resulting in τ^*, where $K^* = \mathsf{Decap}(sk, c^*)$ for a freshly generated sk. By a reduction $\mathcal{R}_{\text{UNPRE}}$ to the UNPRE property of the KEM scheme that is essentially identical to the one in Section 5.3, we have

$$\mathbf{Dist}_{\mathcal{Z}}^{10,11} \leq \mathbf{Adv}^{\text{UNPRE}}_{\mathcal{R}_{\text{UNPRE}}}.$$

Change in Game 13. We first state the new Game 13, and then make the argument.

Game 13: In the case where $\neg(\mathsf{pw} = \mathsf{pw}' \wedge (s^*, T^*) = (s, T) \wedge (c^*, \tau^*) = (c, \tau'))$, find $\mathcal{A}$'s $H_{\mathsf{tag}}(s, T, K^*)$ query resulting in τ^*. If there is none or more than one, set $SK := \perp$. Otherwise there is a unique such $H_{\mathsf{tag}}(s, T, K^*)$ query, and

- For each $\langle \mathsf{pw}^*, pk^*, sk^* \rangle$ record, compute $K := \mathsf{Decap}(sk^*, c^*)$ and check if $K^* = K$. If there is more than one such record, abort.
- If there is no such $\langle \mathsf{pw}^*, pk^*, sk^* \rangle$ record, set $SK := \perp$.
- If there is a unique such $\langle \mathsf{pw}^*, pk^*, sk^* \rangle$ record, but $\mathsf{pw}^* \neq \mathsf{pw}$, also set $SK := \perp$.
- If there is a unique such $\langle \mathsf{pw}^*, pk^*, sk^* \rangle$ record, and $\mathsf{pw}^* = \mathsf{pw}$, set $SK := H_{\mathsf{key}}(K)$.

If $\mathcal{A}$ makes no $H_{\mathsf{tag}}(s, T, \star)$ query resulting in τ^*, then both Game 12 and Game 13 set $SK := \perp$. The probability that $\mathcal{A}$ makes two different $H_{\mathsf{tag}}(s, T, \star)$ queries both resulting in τ^* is upper-bounded by $q_{\mathsf{tag}}(q_{\mathsf{tag}} - 1)/2^{2n+1}$. (This part does not change from the argument under the old Game 13.) Below we assume that $\mathcal{A}$ makes a unique $H_{\mathsf{tag}}(s, T, K^*)$ query resulting in τ^*:

- We first upper-bound the probability that there is more than one $\langle \mathsf{pw}^*, pk^*, sk^* \rangle$ record with $\mathsf{Decap}(sk^*, c^*) = K^*$. Note that in each such record, $(pk^*, sk^*) \leftarrow \mathsf{Gen}$ is freshly generated, so a straightforward reduction to the CR property of the KEM scheme, $\mathcal{R}_{\text{CR1}}$, shows that the probability that the same c^* decapsulates to the same K^* under two freshly generated sk^* is upper-bounded by $\binom{q_2}{2} \cdot \mathbf{Adv}^{\text{CR}}_{\mathcal{R}_{\text{CR1}}}$.
- If there is no $\langle \mathsf{pw}^*, pk^*, sk^* \rangle$ record with $\mathsf{Decap}(sk^*, c^*) = K^*$, then both Game 12 and Game 13 set $SK := \perp$.
- Next, suppose there is a unique $\langle \mathsf{pw}^*, pk^*, sk^* \rangle$ record with $\mathsf{Decap}(sk^*, c^*) = K^*$, but $\mathsf{pw}^* \neq \mathsf{pw}$. If there is no $\langle \mathsf{pw}, pk, sk \rangle$ record, then both Game 12 and Game 13 set $SK := \perp$. If there is a $\langle \mathsf{pw}, pk, sk \rangle$ record, then $(pk, sk) \leftarrow \mathsf{Gen}$ is freshly generated, so a straightforward reduction to the CR property of the KEM scheme, $\mathcal{R}_{\text{CR2}}$, shows that the probability that the same c^* decapsulates to the same K^* under both sk and sk^* is upper-bounded by $q_2 \cdot \mathbf{Adv}^{\text{CR}}_{\mathcal{R}_{\text{CR2}}}$. If this does not happen, then again both Game 12 and Game 13 set $SK := \perp$.

- Finally, if there is a unique $\langle \mathsf{pw}^*, pk^*, sk^* \rangle$ record, and $\mathsf{pw}^* = \mathsf{pw}$, then both Game 12 and Game 13 set $SK := H_{\mathsf{key}}(K)$.

In sum, we have

$$\mathbf{Dist}_{\mathcal{Z}}^{12,13} \leq \frac{q_2(q_2-1)}{2} \cdot \mathbf{Adv}^{\mathrm{CR}}_{\mathcal{R}_{\mathrm{CR1}}} + q_2 \cdot \mathbf{Adv}^{\mathrm{CR}}_{\mathcal{R}_{\mathrm{CR2}}} + \frac{q_{\mathsf{tag}}(q_{\mathsf{tag}}-1)}{2^{2n+1}}.$$

Overall, we conclude that the "only hash (s, T)" version is secure under the CBIND, UNPRE, KBIND and CR properties of the KEM scheme. Since UNPRE is implied by CR per Lemma 3, we can drop UNPRE and the protocol is secure under CBIND, KBIND and CR.

The "hash (s, T) and c" version. If we define τ as $H_{\mathsf{tag}}(s, T, c, K)$, then the second bullet of Claim 4 is not needed, and the protocol can be proven secure without the CBIND property of the KEM scheme.

We claim that the KBIND property can also be removed. KBIND appears in the first bullet of Claim 4. We now have $\mathsf{pw} \neq \mathsf{pw}' \wedge c^* = c$, so in the corresponding reduction $\mathcal{A}$ is forced to send (c^*, τ^*) for $c^* = c$. That is, instead of reducing to KBIND, we only need to reduce to the property in Lemma 2, which is implied by UNI-PK and OW. Overall we only need CR in this version.

6 Future Directions

While we aim to provide a comprehensive and thorough analysis of all OEKE-2F variants, there are, unfortunately (or perhaps fortunately?), still a number of open questions stemmed from this work. Below we mention four topics for future work.

Minimal assumptions for OEKE-2F. We did not prove any results on whether the KEM security properties used in this work are *minimal*, i.e., whether some weaker properties suffice for OEKE-2F security. It appears to us that the properties in Section 2.1 are minimal, but it remains to be formally proven that the security of OEKE (even the "hash everything" version) implies the KEM properties in Section 2.1, i.e., any PPT adversary that breaks any of these properties can be used to build an adversary that breaks the UC-security of OEKE.

On the other hand, with some thoughts one can realize that some of the KEM properties in Section 2.2 are *not* minimal, and certain weaker properties suffice. Take CBIND as an example: even if an adversary can break CBIND of the underlying KEM scheme, as long as this behavior could be "caught" by the simulator, the simulation would still go through. Thus, roughly speaking CBIND can be weakened to the following property: there exists a PPT simulator $\mathcal{S}$ such that for any PPT adversary $\mathcal{A}$ in the CBIND game (Definition 7), $\mathcal{S}$ outputs a bit indicating whether $\mathcal{A}$ wins with all but negligible accuracy. ($\mathcal{S}$ does not know the secret key sk.) A similar weakening can be done for KBIND. Furthermore, it might be possible to remove the KEM key from the adversary's

input in CBIND (Remark 4), and remove one of the two KEM public keys as well as the KEM key from the adversary's input in KBIND (Remark 6).

Regarding UNPRE and CR, these two properties appear minimal as they correspond to "actual" attacks on OEKE — except for UNPRE when pw is hashed but pk is not, i.e., τ is defined as $H_{\mathsf{tag}}(\mathsf{pw}, s, T, K)$ (or $H_{\mathsf{tag}}(\mathsf{pw}, s, T, c, K)$, $H_{\mathsf{tag}}(\mathsf{pw}, K)$ or $H_{\mathsf{tag}}(\mathsf{pw}, c, K)$). Indeed, this is a confusing situation: From Section 1.1 it should be clear that UNPRE is not needed in these OEKE variants, but the actual security analysis in Section 5.3 does assume UNPRE. As discussed after Definition 8, the UC simulator is not able to tell if the adversary is performing an actual attack by breaking UNPRE or sending some random junk, so it is unclear how the simulator should proceed. This suggests that a weaker variant of UNPRE, where a simulator can tell whether the adversary has won or not (similar to the weakening of CBIND and KBIND above), should suffice.

OEKE-2F security in the game-based setting. A natural question is whether we could rely on weaker KEM security properties if we only require game-based security for OEKE-2F. The standard game-based security notion for PAKE is the one in [BPR00] (henceforth BPR). One important difference between the BPR and UC security notions is that BPR does not require forward secrecy. From the discussion in Section 2.1, it seems that some mechanisms in the OW and ANO games (e.g., giving the public key pk to the adversary) are designed for the sole purpose of achieving forward secrecy, so some weaker versions of OW-1PCA and ANO-1PCA should suffice for the BPR security of OEKE-2F.

Furthermore, it seems that the CBIND and KBIND properties do not correspond to any "real" attack on OEKE-2F (see Footnote 10), and are needed only for the simulation to go through. Therefore, it is possible that they can be removed in game-based security of (any variant of) OEKE-2F.

On the other hand, it is unclear how significant a game-based security analysis of OEKE-2F would be. Forward secrecy is considered a standard property for PAKE nowadays, and the UC notion has superseded the game-based one(s) in general. As such, this direction does not seem to be the most urgent in the analysis of OEKE-2F.

Extension to EKE. This entire work is on OEKE only, and one might wonder including/excluding certain items from the hash would make a difference in EKE. Note that EKE does not have the authenticator τ' to begin with, so the only question is what should be included in H_{key} inputs (that derive the session key SK). [JRX25, Appendix A.4] shows that the second protocol message needs to be included and the first protocol message does not. It is not entirely clear whether hashing the password would weaken the security requirements on KEM: from the EKE security proof in [JRX25] it seems that hashing the password does not help, but a formal analysis is open.

Extension to Tempo. Finally, we mention the future direction of the greatest practical significance. The current description and security analysis of the Tempo protocol [ABJ26, Figure 2] assume the "hash everything" version. Given how similar the security proof for Tempo is to the proof for OEKE-2F, we speculate that the same optimization — removing certain items from the ROs while relying on mildly stronger security properties of the underlying KEM scheme — can be made in Tempo, which would further protect the protocol against side-channel attacks.

Acknowledgement. I thank Afonso Arriaga for pointing me to [AMO+26], which in turn helped me realize an error in my proof of Lemma 3 (see Footnote 32).

References

When we cite (say) [ABR01, Definition 9], we always refer to the full version (if available) rather than the proceedings version.

[ABB+20] Michel Abdalla, Manuel Barbosa, Tatiana Bradley, Stanislaw Jarecki, Jonathan Katz, and Jiayu Xu. Universally composable relaxed password authenticated key exchange. In *CRYPTO 2020*, pages 278–307, 2020. Full version available at https://eprint.iacr.org/2020/320.pdf.

[ABJ25] Afonso Arriaga, Manuel Barbosa, and Stanislaw Jarecki. NoIC: PAKE from KEM without ideal ciphers, 2025. https://eprint.iacr.org/2025/231.pdf.

[ABJ26] Afonso Arriaga, Manuel Barbosa, and Stanislaw Jarecki. Tempo: ML-KEM to PAKE compiler resilient to timing attacks. *TCHES*, 2026(3), 2026. To appear; available at https://eprint.iacr.org/2025/1399.pdf.

[ABP15] Michel Abdalla, Fabrice Benhamouda, and David Pointcheval. Public-key encryption indistinguishable under plaintext-checkable attacks. In *PKC 2015*, pages 332–352, 2015. Full version available at https://eprint.iacr.org/2014/609.pdf.

[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. Full version, titled "DHIES: An encryption scheme based on the Diffie-Hellman Problem", available at https://cseweb.ucsd.edu/~mihir/papers/dhies.pdf.

[AHH+24] Nouri Alnahawi, Kathrin Hövelmanns, Andreas Hülsing, Silvia Ritsch, and Alexander Wiesmaier. Towards post-quantum secure PAKE - A tight security proof for OCAKE in the BPR model.

In *CANS 2024*, pages 191–212, 2024. Full versoin available at https://eprint.iacr.org/2023/1368.pdf.

[AMO+26] Afonso Arriaga, David Mestel, Jan Oupický, Peter Browne Rønne, and Marjan Škrobot. HIC is all you need: Practical post-quantum password-authenticated public-key encryption. In *CT-RSA 2026*, pages 127–154, 2026. Full version available at https://eprint.iacr.org/2026/020.pdf.

[AP05] Michel Abdalla and David Pointcheval. Simple password-based encrypted key exchange protocols. In *CT-RSA 2005*, pages 191–208, 2005. Full version available at https://www.di.ens.fr/~mabdalla/papers/AbPo05a-letter.pdf.

[BCP03] Emmanuel Bresson, Olivier Chevassut, and David Pointcheval. Security proofs for an efficient password-based key exchange. In *CCS 2003*, pages 241–250, 2003. Full version available at https://eprint.iacr.org/2002/192.ps.

[BCP04] Emmanuel Bresson, Olivier Chevassut, and David Pointcheval. New security results on encrypted key exchange. In *PKC 2004*, pages 145–158, 2004. Full version available at https://www.di.ens.fr/david.pointcheval/Documents/Papers/2004_pkc.pdf.

[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. Full version available at https://eprint.iacr.org/2023/470.pdf.

[BDZ03] Feng Bao, Robert H. Deng, and Huafei Zhu. Variations of Diffie-Hellman problem. In *ICICS 2003*, pages 301–312, 2003.

[Bla12] Bruno Blanchet. Automatically verified mechanized proof of one-encryption key exchange. In *CSF 2012*, pages 325–339, 2012. Full version available at https://eprint.iacr.org/2012/173.ps.

[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. Available at https://www.cs.columbia.edu/~smb/papers/neke.pdf.

[BPR00] Mihir Bellare, David Pointcheval, and Phillip Rogaway. Authenticated key exchange secure against dictionary attacks. In *EUROCRYPT 2000*, pages 139–155, 2000. Full version available at https://eprint.iacr.org/2000/014.ps.

[BS23] Dan Boneh and Victor Shoup. A graduate course in applied cryptography (version 0.6), 2023. https://toc.cryptobook.us/book.pdf.

[Can01] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *FOCS 2001*, pages 136–145, 2001. Full version available at https://eprint.iacr.org/2000/067.pdf.

[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. Full version available at https://eprint.iacr.org/2005/196.ps.

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

[DS16] Yuanxi Dai and John P. Steinberger. Indifferentiability of 8-round Feistel networks. In *CRYPTO 2016*, pages 95–120, 2016. Full version available at https://eprint.iacr.org/2015/1069.pdf.

[GMP22] Paul Grubbs, Varun Maram, and Kenneth G. Paterson. Anonymous, robust post-quantum public key encryption. In *EUROCRYPT 2022*, pages 402–432, 2022. Full version available at https://eprint.iacr.org/2021/708.pdf.

[HHKR25] Kathrin Hövelmanns, Andreas Hölsing, Mikhail Kudinov, and Silvia Ritsch. CAKE requires programming - On the provable post-quantum security of (O)CAKE, 2025. https://eprint.iacr.org/2025/458.pdf.

[HL19] Björn Haase and Benoît Labrique. AuCPace: Efficient verifier-based PAKE protocol tailored for the IIoT. *TCHES*, 2019(2), 2019. Full version available at https://eprint.iacr.org/2018/286.pdf.

[JRX25] Jake Januzelli, Lawrence Roy, and Jiayu Xu. Under what conditions is encrypted key exchange actually secure? In *EUROCRYPT 2025*, pages 451–481, 2025. Full version available at https://eprint.iacr.org/2024/324.pdf.

[LLMT25] Chengyu Lin, Zeyu Liu, Peihan Miao, and Max Tromanhauser. Finding balance in unbalanced PSI: A new construction from single-server PIR. *CiC*, 2(1):27, 2025. Available at https://cic.iacr.org/p/2/1/27/pdf.

[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. Full version available at https://eprint.iacr.org/2020/1043.pdf.

[SGJ23] 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. Full version available at https://eprint.iacr.org/2023/295.pdf.

[VJW26] Jelle Vos, Stanislaw Jarecki, and Christopher A. Wood. Hybrid post-quantum password authenticated key exchange, 2026. https://datatracker.ietf.org/doc/draft-vos-cfrg-pqpake.

[Xu25] Jiayu Xu. UC-security of encrypted key exchange: A tutorial, 2025. https://eprint.iacr.org/2025/237.pdf.

A Comparison Between Our Simulator and the One in [ABJ25]

Our simulator in Section 4.1.2 is, by and large, the same as the one in [ABJ25, Fig.5 and 6], although we completely rewrite it to improve readability. We first summarize some of the notational differences in Table 7.

	our notation	notation in [ABJ25]
protocol parties	P, P'	A, B
simulated P-to-P' message	(s, T)	$(\overline{s}, \overline{T})$
adversarial P-to-P' message	(s^*, T^*)	(s, T)
$\mathcal{A}$'s guess of P''s password	$(\mathsf{pw}')^*$	pw^*
KEM public key corresponding to correct password guess on P'	pk'	pk^*
simulated P'-to-P message	(c, τ')	(c, tag)
adversarial P'-to-P message	(c^*, τ^*)	(c, tag)
$\mathcal{A}$'s guess of P's password	pw^*	pw^*

Table 7: Comparison of notations between our simulator and the one in [ABJ25]

The simulator in [ABJ25] has more bookkeeping and aborts more frequently. In particular, we completely remove their record forwS, and when detecting whether an adversarial P-to-P' message contains more than one password guess (causing the simulator to abort), we only check the relevant H_2 queries, whereas they check *all* H_2 queries — including those that are completely unrelated to the attack in the P-to-P' direction. For a more detailed explanation, see the comment after Game 15 in Section 4.2.

The only (more or less) essential difference lies in the red text in Figure 6: in the case where the adversary makes an incorrect password guess on P', we compute c from a uniformly random KEM public key pk', whereas they compute c from an honestly generated pk'. While both methods work, our method simplifies the reduction; this is further explained in the comment after Game 17 in Section 4.2, item (2).

B Security Analyses of Other Versions (II): Removing (s, T) from the Hash

B.1 The "Hash pw and pk" Version

We start from the version where τ is defined as $H_{\mathsf{tag}}(\mathsf{pw}, pk, K)$, i.e., (s, T) and c are removed from the hash compared to the "hash everything" version. This would best be compared with the protocol (and its security proof) in Section 5.3, where τ is defined as $H_{\mathsf{tag}}(\mathsf{pw}, pk, s, T, K)$. Removing (s, T) from

the hash introduces a new case that the simulator must take into consideration, which we start from.

B.1.1 Changes to the Simulator

The new situation emerges when the adversary $\mathcal{A}$ decrypts the P-to-P' message (s, T) to recover $pk := \mathsf{2F}^{-1}(\mathsf{pw}, (s, T))$, and re-encrypts the same pk under the same pw to get $(s^*, T^*) \leftarrow \mathsf{2F}(\mathsf{pw}, pk)$. Note that unlike IC, *the* 2F *encryption is randomized*, so $\mathcal{A}$'s re-encryption likely results in a different $(s^*, T^*) \neq (s, T)$. Then $\mathcal{A}$ sends (s^*, T^*) to P', and forwards the P'-to-P message (c, τ') without modification. We show this whole process in Figure 13.

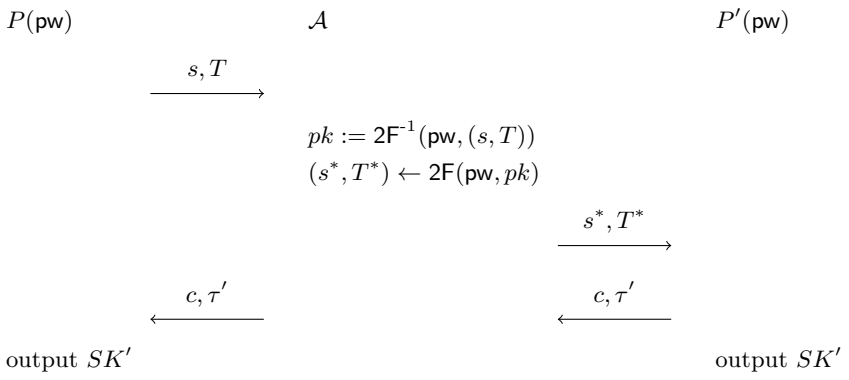

Figure 13: New "attack" on the "hash pw and pk" version

One can intuitively tell that this does not constitute an actual attack, since the only thing $\mathcal{A}$ achieves is to modify some protocol messages and cause P and P' to output the same session key — which $\mathcal{A}$ *cannot* compute.[28] What $\mathcal{A}$ essentially does is to run the 2F encryption process using a fresh randomness, yielding another (s^*, T^*) which has the same effect on the P' side as the honest (s, T); in this way, $\mathcal{A}$ is "effectively" passive while actually modifying the P-to-P' message. (We stress again that this "attack" critically relies on 2F being randomized, and does not work anymore if we replace it with an IC. However, all subsequent "attacks" — in the setting where we further remove pw and/or pk from H_{tag} inputs — work even with IC.)

This example vividly demonstrates why removing (s, T) from the hash sometimes does not require more properties of the underlying KEM scheme, but complicates the security proof. Since this "attack" requires $\mathcal{A}$ to decrypt and re-encrypt via some RO queries, the simulator can catch $\mathcal{A}$ doing so (and can extract the password along the way), and no additional properties on the KEM level are needed. Still, this scenario is not covered by the previous simulator,

[28] If anything, such an $\mathcal{A}$ is "dumber" than an adversary who performs a regular online attack, since $\mathcal{A}$ who knows the correct password could have learned the two parties' session keys but wastes such an opportunity.

and the simulator has to create a new case dedicated to this specific scenario. In more detail, the scenario in Figure 13 should be simulated as follows: When $\mathcal{A}$ decrypts (s, T) under pw, the simulator $\mathcal{S}$ records $\langle \mathsf{pw}, pk, sk \rangle$ (although there might be a large number of such decryption attempts, and $\mathcal{S}$ does not know which is the "right" one at this point). Then when $\mathcal{A}$ sends (s^*, T^*) to P', $\mathcal{S}$ extracts the "right" pw, which allows $\mathcal{S}$ to compromise P''s session. Later, when $\mathcal{A}$ forwards (c^*, τ'), $\mathcal{S}$ can tell that $\mathcal{A}$ is performing the aforementioned "attack", and can use the same pw to compromise P's session.

Remark 10. Looking ahead, all subsequent "attacks" follow a similar pattern:

- In the first protocol message, $\mathcal{A}$ decrypts (s, T) and re-encrypts to some other (s^*, T^*), and
- In the second protocol message, $\mathcal{A}$ forwards τ' (but may or may not modify c).

If we include (s, T) in H_{tag} inputs, this will immediately result in P rejecting, since P's check will fail. However, without (s, T) in the hash, we have to take this scenario into account. The difference from what's in this section is that if we additionally remove pw and/or pk from the hash, then $\mathcal{A}$ can re-encrypt pk^* under pw', where pw' is equal or not equal to pw, and pk^* is equal or not equal to pk. Hence, Figure 13 is a special case of all subsequent "attacks".

We show the new simulator in Figure 14. (Only the steps after receiving (c^*, τ^*) are included.)

Remark 11. We briefly mention another possible simulation strategy. Referring to Figure 13, computing $pk := \mathsf{2F}^{-1}(\mathsf{pw}, (s, T))$ requires $\mathcal{A}$ to query $H_2(\mathsf{pw}, T)$, at which time the simulator $\mathcal{S}$ stores $\langle \mathsf{pw}, pk, sk \rangle$. Later when $\mathcal{S}$ simulates the P'-to-P message, it needs to compute $(c, K) \leftarrow \mathsf{Encap}(pk)$ and $\tau' := H_{\mathsf{tag}}(\mathsf{pw}, pk, K)$ (see step 2(b)(ii) of Figure 6). Therefore, in Figure 14 we could remove step 2, and instead in step 3 find $\mathcal{A}$ *or* $\mathcal{S}$ *itself*'s $H_{\mathsf{tag}}(\mathsf{pw}^*, pk^*, K^*)$ query resulting in τ^*, which will still result in $\mathcal{S}$ finding pw and sending $(\mathsf{TestPwd}, sid, P, \mathsf{pw})$ to $\mathcal{F}$. This is essentially what the simulator in [JRX25] does, and yields a simpler simulator. We choose to present our simulator in the more complicated way because it explicitly highlights the "attack" in Figure 13 and thus provides a clearer comparison with the "hash everything" version of the protocol.

The same remark goes for subsequent simulators in this section.

P's session key

On (c^*, τ^*) from $\mathcal{F}$, simulate P's output as follows:

1. If $(s^*, T^*) = (s, T) \wedge (c^*, \tau^*) = (c, \tau')$, send $(\mathsf{NewKey}, sid, P, 0^n)$ to $\mathcal{F}$. // $\mathcal{A}$ is passive
2. If $(s^*, T^*) \neq (s, T) \wedge (c^*, \tau^*) = (c, \tau')$, and there has been a $(\mathsf{TestPwd}, sid, P', \mathsf{pw}')$ command to $\mathcal{F}$ resulting in "correct guess" (i.e., $\mathcal{S}$ previously entered step 2(b)(ii) in Figure 7), there must be a pk' and an SK' defined at that time.
   (a) Retrieve record $\langle \mathsf{pw}', pk, \star \rangle$. If there is no such record, go to step 3. // $\mathcal{A}$ does not decrypt $pk := \mathsf{2F}^{-1}(\mathsf{pw}, (s, T))$
   (b) If $pk \neq pk'$, go to step 3. // $\mathcal{A}$ decrypts (s, T) and gets pk, but then re-encrypts $(s^*, T^*) \leftarrow \mathsf{2F}(\mathsf{pw}, pk')$ for another pk'
   (c) Send $(\mathsf{TestPwd}, sid, P, \mathsf{pw}')$ to $\mathcal{F}$.
      i. If $\mathcal{F}$ replies with "wrong guess", send $(\mathsf{NewKey}, sid, P, 0^n)$ to $\mathcal{F}$. // P and P''s passwords are different
      ii. If $\mathcal{F}$ replies with "correct guess", send $(\mathsf{NewKey}, sid, P, SK')$ to $\mathcal{F}$.
3. Otherwise find $\mathcal{A}$'s $H_{\mathsf{tag}}(\mathsf{pw}^*, pk^*, K^*)$ query resulting in τ^*.
   (a) If there is no such query or more than one such query, send $(\mathsf{TestPwd}, sid, P, \perp)$ and then $(\mathsf{NewKey}, sid, P, 0^n)$ to $\mathcal{F}$.
   (b) If there is exactly one such query, retrieve record $\langle \mathsf{pw}^*, \star, sk^* \rangle$. If there is no such record, send $(\mathsf{TestPwd}, sid, P, \perp)$ and then $(\mathsf{NewKey}, sid, P, 0^n)$ to $\mathcal{F}$.
   (c) Compute $K := \mathsf{Decap}(sk^*, c^*)$. If $K^* \neq K$, send $(\mathsf{TestPwd}, sid, P', \perp)$ and then $(\mathsf{NewKey}, sid, P, 0^n)$ to $\mathcal{F}$.
   (d) Send $(\mathsf{TestPwd}, sid, P, \mathsf{pw}^*)$ to $\mathcal{F}$.
      i. If $\mathcal{F}$ replies with "wrong guess", send $(\mathsf{NewKey}, sid, P, 0^n)$ to $\mathcal{F}$.
      ii. If $\mathcal{F}$ replies with "correct guess", compute $SK := H_{\mathsf{key}}(K)$ and send $(\mathsf{NewKey}, sid, P, SK)$ to $\mathcal{F}$.

Figure 14: UC simulator $\mathcal{S}$ for the "hash pw and pk" variant of OEKE. Only the paragraph that needs essential change (shown in blue) is included

B.1.2 Changes in Game Hops

Note that Claim 1 in Game 2 does not hold anymore, exactly due to what the adversary can do in Figure 13 (P does not reject even though $\mathcal{A}$ is not passive and $\mathcal{A}$ makes no H_{tag} query whatsoever). As such, Game 2 needs to change essentially. Since Game 2 is almost at the beginning of the sequence of games, this affects some subsequent games and we need to proceed carefully. The trick

is to isolate the specific case in Figure 13, i.e., the case that makes Claim 1 fail; in this case we need to additionally check P''s H_{tag} query, whereas in all other cases we can proceed as before. In this way we minimize the effect of the change in Game 2.

We start from describing the new Game 2. Most of the subsequent argument is clerical work verifying that the previous argument still goes through; the parts that need to essentially change are shown in blue.

Game 2: In the case where $\mathsf{pw} \neq \mathsf{pw}' \vee c^* \neq c$, find $\mathcal{A}$'s $H_{\mathsf{tag}}(\mathsf{pw}, pk, K^*)$ query resulting in τ^*. If there is none or more than one, set $SK := \bot$. If there is a unique such query, do the following:

- If $K^* = K$, set $SK := H_{\mathsf{key}}(K)$ (i.e., there is no change from Game 1).
- If $K^* \neq K$, set $SK := \bot$.

Furthermore, in the case where $\mathsf{pw} = \mathsf{pw}' \wedge (s^*, T^*) \neq (s, T) \wedge c^* = c$, do the same as above, except that we now find $\mathcal{A}$ *or* P''s $H_{\mathsf{tag}}(\mathsf{pw}, pk, K^*)$ query resulting in τ^*.

The difference between the new Game 2 and the old one is that we now have two separate cases:

1. $\mathsf{pw} \neq \mathsf{pw}' \vee c^* \neq c$;
2. $\mathsf{pw} = \mathsf{pw}' \wedge (s^*, T^*) \neq (s, T) \wedge c^* = c$.

These two cases combined yield $\neg(\mathsf{pw} = \mathsf{pw}' \wedge (s^*, T^*) = (s, T) \wedge c^* = c)$, which is identical to the condition in the old Game 2 (i.e., they cover all situations other than the one addressed in Game 1). In case (1) we proceed as in the old Game 2, whereas in case (2) we cannot due to the "attack" in Figure 13; instead, in case (2) we need to include P''s H_{tag} query as if it were $\mathcal{A}$'s.

Claim 5. *Suppose the CBIND property holds for the KEM scheme. In case (1) above, the probability that P' queries $H_{\mathsf{tag}}(\mathsf{pw}, pk, K)$ is negligible.*

Proof. P' makes a single H_{tag} query, on (pw', pk', K'). If $\mathsf{pw} \neq \mathsf{pw}'$, then of course P''s query is not $H_{\mathsf{tag}}(\mathsf{pw}, pk, K)$. Now assume $\mathsf{pw} = \mathsf{pw}'$. This implies $c^* \neq c$ by the condition of the claim.

We now claim that $pk = pk' \wedge K = K'$ happens with negligible probability. Suppose $pk = pk'$. Then we can reduce the event $K = K'$ to the CBIND property of the KEM scheme, via a reduction $\mathbf{Adv}^{\mathrm{CBIND}}_{\mathcal{R}_{\mathrm{CBIND}}}$ similar to that in the proof of Claim 2. We have $\Pr[pk = pk' \wedge K = K'] \leq \mathbf{Adv}^{\mathrm{CBIND}}_{\mathcal{R}_{\mathrm{CBIND}}}$.

If $pk \neq pk'$ or $K \neq K'$ then of course P''s query is not $H_{\mathsf{tag}}(\mathsf{pw}, pk, K)$. We have

$$\Pr[P' \text{ queries } H_{\mathsf{tag}}(\mathsf{pw}, pk, K)] \leq \mathbf{Adv}^{\mathrm{CBIND}}_{\mathcal{R}_{\mathrm{CBIND}}},$$

which yields the claim. □

The remaining argument is essentially identical to that under the old Game 2.

For subsequent games, if they only involve case (1) in Game 2 then they do not need to change, since the new Game 2 does not make any change in this case. However, if a game involves case (2) then some changes need to be made, since the new Game 2 additionally checks P''s H_{tag} queries, which a reduction might not be able to do.

Games 3–4: These two games deal with the case where $\mathsf{pw} = \mathsf{pw}' \wedge (s^*, T^*) = (s, T)$. This falls into either the case in Game 1 (if $c^* = c$) or case (1) in Game 2 (if $c^* \neq c$); either way, it does not enter case (2) in Game 2, so the reductions still go through: they abort if $\mathsf{pw} = \mathsf{pw}' \wedge (s^*, T^*) = (s, T)$ does not hold, and if $\mathsf{pw} = \mathsf{pw}' \wedge (s^*, T^*) = (s, T)$, they can still simulate the games by checking $\mathcal{A}$'s H_{tag} queries. Crucially, the reductions do not need to check P''s H_{tag} query (which they cannot do), since in this case we have defined Game 2 so that it only checks $\mathcal{A}$'s H_{tag} queries.

Game 5: This game involves a reduction to the UNI-PK property of the KEM scheme. The reduction works essentially because sk' is not used anywhere in the game (so the reduction can embed its challenge as pk' without knowing sk'), and this fact still holds even though we need to additionally check P''s H_{tag} queries in some cases now. So, this reduction still goes through.

Games 6–8: These three games deal with the case where $\mathsf{pw} \neq \mathsf{pw}' \wedge (s^*, T^*) = (s, T)$. This falls into case (1) in Game 2, so the reductions in Games 6 and 7 still go through; Game 8 involves a statistical argument which does not need to change either.

Games 9–10: These two games involve statistical arguments only, and they have nothing to do with whether the games need to check P''s H_{tag} queries.

Game 11: This game hop goes through for essentially the same reason as in Section 4.2. The difference is that now in case (2), Game 10 and Game 11 differ if $\mathcal{A}$ does not make an $H_2(\mathsf{pw}, T)$ query, but $\mathcal{A}$ *or* P' makes an $H_{\mathsf{tag}}(\mathsf{pw}, pk, K^*)$ query resulting in τ^*. However, in vase (2) pk is freshly generated by P, independent of *both* $\mathcal{A}$ and P''s views. Thus, the distinguishing advantage between Game 10 and Game 11 is still upper-bounded by p_{guess}.

Game 12: This game should read as follows now: In the case where $\mathsf{pw} = \mathsf{pw}' \wedge (s^*, T^*) = (s, T) \wedge c^* = c \wedge \tau^* \neq \tau'$, do what's in the $\mathsf{pw} \neq \mathsf{pw}' \vee c^* \neq c$ case (i.e., case (1) in Game 2).

This game only involves case (1) in Game 2, i.e., it checks $\mathcal{A}$'s H_{tag} queries *but does not check P''s H_{tag} query*. Therefore, this game hop has nothing to do with the change in the new Game 2, and the previous argument still goes through.

Game 13: This game should now extract pw^* from $\mathcal{A}$ *or* P''s H_{tag} query, if case (2) in Game 2 happens. For completeness, we include a full description of the game.

In the case where $\mathsf{pw} \neq \mathsf{pw}' \vee c^* \neq c \vee (\mathsf{pw} = \mathsf{pw}' \wedge (s^*, T^*) = (s, T) \wedge c^* = c \wedge \tau^* \neq \tau')$, find $\mathcal{A}$'s $H_{\mathsf{tag}}(\mathsf{pw}^*, pk^*, K^*)$ query resulting in τ^*. If there is none or more than one, set $SK := \perp$. Otherwise there is a unique such $H_{\mathsf{tag}}(\mathsf{pw}^*, pk^*, K^*)$ query, and

- If $\mathsf{pw}^* \neq \mathsf{pw}$, set $SK := \perp$.
- If $\mathsf{pw}^* = \mathsf{pw}$, there is no change from Game 12.

Furthermore, in the case where $\mathsf{pw} = \mathsf{pw}' \wedge (s^*, T^*) \neq (s, T) \wedge c^* = c$ (i.e., case (2) in Game 2), do the same as above, except that we now find $\mathcal{A}$ *or* P''s $H_{\mathsf{tag}}(\mathsf{pw}^*, pk^*, K^*)$ query resulting in τ^*.

The previous statistical argument is based on ruling out collisions in H_{tag}. Here we can still rule out such collisions, except that the probability of a collision is $q_{\mathsf{tag}}(q_{\mathsf{tag}} + 1)/2$ rather than $q_{\mathsf{tag}}(q_{\mathsf{tag}} - 1)/2$ (since we may additionally count the H_{tag} query by P'). The rest of the argument does not change.

Games 14–16: The statistical arguments for these three games do not need to change.

New Game I: Now that $(\mathsf{pw}')^*$ — the password guess extracted from (s^*, T^*) — is defined (in Game 15), we can finally add (part of) what's in step 2 of Figure 14. Concretely, we change the game as follows:

In the case where $(s^*, T^*) \neq (s, T) \wedge (\mathsf{pw}')^* = \mathsf{pw} = \mathsf{pw}' \wedge c^* = c$, there is a record $\langle \mathsf{pw}, pk, \star \rangle$, and $pk = pk'$, set $SK := SK'$. If $(\mathsf{pw}')^*$ is undefined, treat it as if $(\mathsf{pw}')^* \neq \mathsf{pw}'$. (In some sense this is the most crucial new game, as it covers the scenario in Figure 13 — which is why we need to make all these changes to begin with. It also corresponds to step 2(c)(ii) in the simulator in Figure 14. We will deal with step 2(c)(i) in new Game III later.)

Since $\mathsf{pw} = \mathsf{pw}' \wedge (s^*, T^*) \neq (s, T) \wedge (c^*, \tau^*) = (c, \tau')$, Game 16 finds $\mathcal{A}$ or P''s H_{tag} query resulting in τ^* (see the description of Game 13). P' makes a single query to H_{tag}, on input (pw', pk', K'); and the query result is $\tau' = \tau^*$. Therefore, Game 16 finds this query, and does the following (again see the description of Game 13):

1. Check if $\mathsf{pw}' = \mathsf{pw}$, which is indeed the case.
2. Retrieve the $\langle \mathsf{pw}, pk, sk \rangle$ record (as what Game 11 dictates).
3. Compute $K := \mathsf{Decap}(sk, c^*)$ and check if $K' = K$. Note that

$$(c, K') \leftarrow \mathsf{Encap}(pk') \text{ on the } P' \text{ side},$$
$$c^* = c,$$
$$K = \mathsf{Decap}(sk, c^*) \text{ on the } P \text{ side},$$

furthermore $pk = pk'$ (per the condition of the game) and (pk, sk) are an honestly generated KEM public/secret key pair (per step 2 above), so by the correctness of the KEM scheme, the $K' \stackrel{?}{=} K$ check passes except with probability $\varepsilon_{\text{correctness}}$.

4. Set $SK := H_{\mathsf{key}}(K) (= H_{\mathsf{key}}(K') = SK')$.

This entire process in Game 16 eventually results in setting $SK := SK'$ except with probability $\varepsilon_{\text{correctness}}$, which exactly matches the new game. We have

$$\mathbf{Dist}_{\mathcal{Z}}^{16,\text{newI}} \leq \varepsilon_{\text{correctness}}.$$

In case (2) in Game 2, we have to check both $\mathcal{A}$ and P''s H_{tag} queries, exactly because of the special scenario in new Game I (i.e., in Figure 13). Now that this case has been singled out and dealt with, we can remove the check of P''s H_{tag} queries.

New Game II: In the case where $\mathsf{pw} = \mathsf{pw}' \wedge (s^*, T^*) \neq (s, T) \wedge c^* = c$ (i.e., case (2) in Game 2), we find $\mathcal{A}$ *or* P''s $H_{\mathsf{tag}}(\mathsf{pw}^*, pk^*, K^*)$ query resulting in τ^* — except that in new Game I's condition, new Game I already shortcuts it to $SK := SK'$ without trying to find any H_{tag} query. In this game, we do not check P''s H_{tag} query anymore, and only check $\mathcal{A}$'s H_{tag} query.

Claim 6. *If* $(\mathsf{pw}')^* \neq \mathsf{pw}'$ *then*

1. *pk and pk' are independent of each other. (pk is the item in record $\langle \mathsf{pw}, pk, sk \rangle$.)*
2. $\Pr[pk = pk'] = 1/|\mathbb{G}|$.

Proof. We have (in the creation of the record $\langle \mathsf{pw}, pk, sk \rangle$)

$$pk = \frac{T}{R} = \frac{T}{H_1(\mathsf{pw}, r)} = \frac{T}{H_1(\mathsf{pw}, s \oplus t)} = \frac{T}{H_1(\mathsf{pw}, s \oplus H_2(\mathsf{pw}, T))},$$

and (in P''s algorithm)

$$pk' = \frac{T^*}{R'} = \frac{T^*}{H_1(\mathsf{pw}, r')} = \frac{T^*}{H_1(\mathsf{pw}, s^* \oplus t')} = \frac{T^*}{H_1(\mathsf{pw}, s^* \oplus H_2(\mathsf{pw}, T^*))}$$

(recall $\mathsf{pw} = \mathsf{pw}'$ by the condition of the game). If $\mathcal{A}$ does not query $H_1(\mathsf{pw}, s^* \oplus H_2(\mathsf{pw}, T^*))$, then of course pk' is independent of pk. Now suppose $\mathcal{A}$ queries $H_1(\mathsf{pw}, s^* \oplus H_2(\mathsf{pw}, T^*))$. Recall that $(\mathsf{pw}')^*$ is defined via types (1) and (2) in Game 15 in Section 4.2:

- If $s^* \oplus H_2(\mathsf{pw}, T^*) = s \oplus H_2(\mathsf{pw}, T)$, then $(\mathsf{pw}')^*$ is set to pw in type (2);
- If the $H_1(\mathsf{pw}, s^* \oplus H_2(\mathsf{pw}, T^*))$ query is made before $\mathcal{A}$'s $H_2(\mathsf{pw}, T^*)$ query, then $(\mathsf{pw}')^*$ is set to pw in type (1).

Since $(\mathsf{pw}')^* \neq \mathsf{pw}'$, the only possibility is that $\mathcal{A}$ makes a *fresh* $H_1(\mathsf{pw}, s^* \oplus H_2(\mathsf{pw}, T^*))$ query *after* its $H_2(\mathsf{pw}, T^*)$ query (in particular, after T^* is fixed). This means that $H_1(\mathsf{pw}, s^* \oplus H_2(\mathsf{pw}, T^*))$ is independent of everything else (in particular, independent of pk), which implies that pk' is uniformly random in $\mathbb{G}$ and independent of pk. So $\Pr[pk = pk'] = 1/|\mathbb{G}|$. □

Going back to the game, there are the following cases:

- If $\tau^* \neq \tau'$, then of course P''s H_{tag} query does not result in τ^* (it results in τ'), so we can safely drop P''s H_{tag} query from the check.
- If $\tau^* = \tau'$, but there is no $\langle \mathsf{pw}, pk, \star \rangle$ record, then both new Game I and new Game II outright set $SK := \bot$, regardless of H_{tag} queries by $\mathcal{A}$ or P'. Thus, the two games are identical.
- If $\tau^* = \tau'$, there is a record $\langle \mathsf{pw}, pk, \star \rangle$, but $(\mathsf{pw}')^* \neq \mathsf{pw}' \vee pk \neq pk'$, then P''s $H_{\mathsf{tag}}(\mathsf{pw}', pk', K')$ query results in τ^*, and we need to argue that this query can be dropped from the check.
  - If $\mathcal{A}$ queries $H_{\mathsf{tag}}(\mathsf{pw}', pk', K')$, then new Game II still finds this query, so new Game I and new Game II are identical.
  - If $\mathcal{A}$ does not query $H_{\mathsf{tag}}(\mathsf{pw}', pk', K')$, then new Game II sets $SK := \bot$. We argue that new Game I also sets $SK := \bot$ except with negligible probability. New Game I finds P''s $H_{\mathsf{tag}}(\mathsf{pw}', pk', K')$ query, and sets $SK := \bot$ if $pk \neq pk'$. But by Claim 6 we have $\Pr[pk = pk'] \leq 1/|\mathbb{G}|$.

The remaining case is that $(\mathsf{pw}')^* = \mathsf{pw}' \wedge \tau^* = \tau'$, there is a record $\langle \mathsf{pw}, pk, \star \rangle$, and $pk = pk'$. The condition of the game additionally requires $\mathsf{pw} = \mathsf{pw}' \wedge (s^*, T^*) \neq (s, T) \wedge c^* = c$. Combined, we get "$(s^*, T^*) \neq (s, T) \wedge (\mathsf{pw}')^* = \mathsf{pw} = \mathsf{pw}' \wedge (c^*, \tau^*) = (c, \tau')$, there is a record $\langle \mathsf{pw}, pk, \star \rangle$, and $pk = pk'$". This is exactly the condition in new Game I, in which case we do not make any change in new Game II. Overall, we conclude that

$$\mathbf{Dist}_{\mathcal{Z}}^{\text{newI,newII}} \leq \frac{1}{|\mathbb{G}|}.$$

Game 17: Note that after new Game II, we never try to find P''s H_{tag} query anymore: in some cases we try to find $\mathcal{A}$'s H_{tag} query, and in other cases we outright set SK to be a certain value or $\bot$. Therefore, the reduction still goes through (it can simulate the P side honestly).[29]

New Game III: In the case where $(s^*, T^*) \neq (s, T) \wedge (\mathsf{pw}')^* = \mathsf{pw}' \neq \mathsf{pw} \wedge (c^*, \tau^*) = (c, \tau')$, there is a record $\langle \mathsf{pw}, pk, \star \rangle$, and $pk = pk'$, set $SK := \bot$. (This corresponds to step 2(c)(i) in Figure 14.)

In this case Game 17 tries to find $\mathcal{A}$'s (unique) $H_{\mathsf{tag}}(\mathsf{pw}^*, pk^*, K^*)$ query resulting in τ^*, and checks if $\mathsf{pw}^* = \mathsf{pw}$ (see Game 13). But $\tau^* = \tau' = H_{\mathsf{tag}}(\mathsf{pw}', pk', K')$, so Game 17 sets $SK := \bot$ because it either fails to find $\mathcal{A}$'s $H_{\mathsf{tag}}(\mathsf{pw}', pk', K')$ query, or it finds such a query but then sets $SK := \bot$ because $\mathsf{pw}^* = \mathsf{pw}' \neq \mathsf{pw}$. Either way, Game 17 matches new Game III which outright sets $SK := \bot$. We have

$$\mathbf{Dist}_{\mathcal{Z}}^{17,\text{newIII}} = 0.$$

[29] New Games I and II — which drop P''s H_{tag} query from the check — have to be inserted before Game 17, since the reduction in Game 17 cannot check P''s H_{tag} query (the query is $H_{\mathsf{tag}}(\mathsf{pw}', pk', K')$, and the reduction does not know K').

Summary. We have made the following essential changes to the games:

1. In the case where $\mathsf{pw} = \mathsf{pw}' \wedge (s^*, T^*) \neq (s, T) \wedge c^* = c$ (case (2) in Game 2), Game 2 tries to find P''s H_{tag} query resulting in τ^* (in addition to $\mathcal{A}$'s query).

2. In the case where $(s^*, T^*) \neq (s, T) \wedge (\mathsf{pw}')^* = \mathsf{pw} = \mathsf{pw}' \wedge (c^*, \tau^*) = (c, \tau')$, there is a record $\langle \mathsf{pw}, pk, \star \rangle$, and $pk = pk'$, new Game I outright sets $SK := SK'$ (without trying to find any H_{tag} query by $\mathcal{A}$ or P').

3. In the case where $\mathsf{pw} = \mathsf{pw}' \wedge (s^*, T^*) \neq (s, T) \wedge c^* = c$, new Game II drops the check on P''s H_{tag} query introduced in Game 2 (except in new Game I's condition, where all checks on H_{tag} queries have already been dropped).

4. In the case where $(s^*, T^*) \neq (s, T) \wedge (\mathsf{pw}')^* = \mathsf{pw}' \neq \mathsf{pw} \wedge (c^*, \tau^*) = (c, \tau')$, there is a record $\langle \mathsf{pw}, pk, \star \rangle$, and $pk = pk'$, new Game III outright sets $SK := \perp$.

Items (1) and (3) introduce the check on P''s H_{tag} query and then drop it, so there is no net change there. To see the net change, we only need to combine items (2) and (4), which say that in the case where $(s^*, T^*) \neq (s, T) \wedge (\mathsf{pw}')^* = \mathsf{pw}' \wedge \tau^* = \tau'$, there is a record $\langle \mathsf{pw}, pk, \star \rangle$, and $pk = pk'$,

- If $(\mathsf{pw}')^* = \mathsf{pw}$, then we set $SK := H_{\mathsf{key}}(K')$ [item (2)].
- If $(\mathsf{pw}')^* \neq \mathsf{pw}$, then we set $SK := \perp$ [item (4)].

This exactly matches step 2 of the simulator in Figure 16, so new Game III is identical to the ideal world. This completes the proof.

The "hash pw, pk and c" version. If we define τ as $H_{\mathsf{tag}}(\mathsf{pw}, pk, c, K)$, then Claim 5 unconditionally holds, and the protocol can be proven secure without the CBIND property of the KEM scheme (i.e., under the same assumptions as in the "hash everything" version).

B.2 The "Hash pw" Version

We now consider the variant where pk is further removed from H_{tag} inputs, i.e., τ is defined as $H_{\mathsf{tag}}(\mathsf{pw}, K)$. This would best be compared with the protocol (and its security proof) in Section 5.3, where τ is defined as $H_{\mathsf{tag}}(\mathsf{pw}, s, T, K)$; as well as the protocol (and its security proof) in Appendix B.1, where τ is defined as $H_{\mathsf{tag}}(\mathsf{pw}, pk, K)$. As we have seen in Appendix B.1, removing (s, T) from the hash introduces a new "re-encryption" case that the simulator must take into consideration. After further removing pk, there is a more general "attack" we need to take into account, which we start from.

B.2.1 Changes to the Simulator

The easiest way to see the new, more general "attack" is to work out a concrete version for OEKE based on the Diffie–Hellman KEM scheme, which we show in Figure 15.

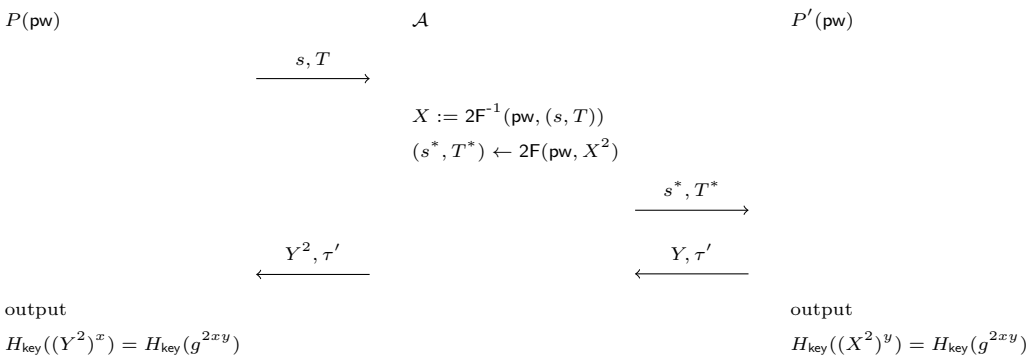

Figure 15: New "attack" on the "hash pw" version, using OEKE based on Diffie–Hellman KEM

In this "attack", the two honest parties' passwords pw are the same. Assuming $\mathcal{A}$ knows it, it can decrypt P's message to recover $X = g^x$, and then re-encrypt X^2 under the same pw; this causes P' to output $SK' = H_{\mathsf{key}}(g^{2xy})$ and send $(Y = g^y, \tau' = H_{\mathsf{tag}}(\mathsf{pw}, g^{2xy}))$. Then $\mathcal{A}$ sends (Y^2, τ') to P, causing P's check to pass and P will also output $H_{\mathsf{key}}(g^{2xy})$. (This is not an issue if pk is included in H_{tag}, since P would compute $\tau = H_{\mathsf{tag}}(\dots, Y^2, \dots) \neq H_{\mathsf{tag}}(\dots, Y, \dots) = \tau'$ unless there is a collision in H_{tag}, and its check $\tau^* \stackrel{?}{=} \tau$ would fail.)

In general, $\mathcal{A}$ can decrypt (s, T) to recover $pk := 2\mathsf{F}^{-1}(\mathsf{pw}, (s, T))$, then re-encrypt some pk' of $\mathcal{A}$'s choice under the same pw' to get $(s^*, T^*) \leftarrow 2\mathsf{F}(\mathsf{pw}, pk')$, and finally forward τ' but manipulate c^* so that P's check will pass. The "attack" in Appendix B.1 is the special case where $pk' = pk$; there, including pk in H_{tag} inputs forces $\mathcal{A}$ not to change pk.

This scenario should be simulated as follows: When $\mathcal{A}$ decrypts (s, T) under pw, $\mathcal{S}$ records $\langle \mathsf{pw}, pk, sk \rangle$ (although there might be a large number of such decryption attempts, and $\mathcal{S}$ does not know which is the "right" one at this point). Then when $\mathcal{A}$ sends (s^*, T^*) to P', $\mathcal{S}$ extracts pw, which allows $\mathcal{S}$ to retrieve the "right" record $\langle \mathsf{pw}, pk, sk \rangle$; $\mathcal{S}$ also computes K' while simulating the P' side. Later, when $\mathcal{A}$ sends (c^*, τ'), $\mathcal{S}$ can tell if $\mathcal{A}$ is performing the aforementioned attack by checking if $\mathsf{Decap}(sk, c^*) = K'$.

We show the new simulator in Figure 16. (Only the steps after receiving (c^*, τ^*) are included.)

P's session key

On (c^*, τ^*) from $\mathcal{F}$, simulate P's output as follows:

1. If $(s^*, T^*) = (s, T) \wedge (c^*, \tau^*) = (c, \tau')$, send $(\mathsf{NewKey}, sid, P, 0^n)$ to $\mathcal{F}$. // $\mathcal{A}$ is passive
2. If $(s^*, T^*) \neq (s, T) \wedge \tau^* = \tau'$, and there has been a $(\mathsf{TestPwd}, sid, P, \mathsf{pw}')$ command to $\mathcal{F}$ resulting in "correct guess" (i.e., $\mathcal{S}$ previously entered step 2(b)(ii) in Figure 7), there must be a K' and an SK' defined at that time.
   (a) Retrieve record $\langle \mathsf{pw}', \star, sk \rangle$. If there is no such record, go to step 3. // $\mathcal{A}$ does not decrypt $pk := 2\mathsf{F}^{-1}(\mathsf{pw}, (s, T))$
   (b) Compute $K := \mathsf{Decap}(sk, c^*)$. If $K \neq K'$, go to step 3. // $\mathcal{A}$ sends an "incorrect" c^*
   (c) Send $(\mathsf{TestPwd}, sid, P, \mathsf{pw}')$ to $\mathcal{F}$.
      i. If $\mathcal{F}$ replies with "wrong guess", send $(\mathsf{NewKey}, sid, P, 0^n)$ to $\mathcal{F}$. // P and P''s passwords are different
      ii. If $\mathcal{F}$ replies with "correct guess", send $(\mathsf{NewKey}, sid, P, SK')$ to $\mathcal{F}$.
3. Otherwise find $\mathcal{A}$'s $H_{\mathsf{tag}}(\mathsf{pw}^*, K^*)$ query resulting in τ^*.
   (a) If there is no such query or more than one such query, send $(\mathsf{TestPwd}, sid, P, \perp)$ and then $(\mathsf{NewKey}, sid, P, 0^n)$ to $\mathcal{F}$.
   (b) If there is exactly one such query, retrieve record $\langle \mathsf{pw}^*, \star, sk^* \rangle$. If there is no such record, send $(\mathsf{TestPwd}, sid, P, \perp)$ and then $(\mathsf{NewKey}, sid, P, 0^n)$ to $\mathcal{F}$.
   (c) Compute $K := \mathsf{Decap}(sk^*, c^*)$. If $K^* \neq K$, send $(\mathsf{TestPwd}, sid, P, \perp)$ and then $(\mathsf{NewKey}, sid, P, 0^n)$ to $\mathcal{F}$.
   (d) Send $(\mathsf{TestPwd}, sid, P, \mathsf{pw}^*)$ to $\mathcal{F}$.
      i. If $\mathcal{F}$ replies with "wrong guess", send $(\mathsf{NewKey}, sid, P, 0^n)$ to $\mathcal{F}$.
      ii. If $\mathcal{F}$ replies with "correct guess", compute $SK := H_{\mathsf{key}}(K)$ and send $(\mathsf{NewKey}, sid, P, SK)$ to $\mathcal{F}$.

Figure 16: UC simulator $\mathcal{S}$ for the "hash pk" variant of OEKE. Only the paragraph that needs essential change (shown in blue) is included

B.2.2 Changes in Game Hops

The new security proof has a very similar structure to that in Appendix B.1.2: split Game 2 into two cases and additionally check P''s H_{tag} queries in the second case; verify that all of Games 3–16 still go through; insert new Games I and II between Game 16 and Game 17; verify that Game 17 still goes through; insert new Game III after Game 17. Below we omit Games 3–17 and only describe Game 2 and the three new games. (In fact even a large chunk of the

arguments for the three new games is very similar to those in Appendix B.1.2, but we describe them in full.)

Game 2: In the case where $\mathsf{pw} \neq \mathsf{pw}' \vee ((s^*, T^*) = (s, T) \wedge c^* \neq c)$, find $\mathcal{A}$'s $H_{\mathsf{tag}}(\mathsf{pw}, K^*)$ query resulting in τ^*. If there is none or more than one, set $SK := \perp$. If there is a unique such query, do the following:

- If $K^* = K$, set $SK := H_{\mathsf{key}}(K)$ (i.e., there is no change from Game 1).
- If $K^* \neq K$, set $SK := \perp$.

Furthermore, in the case where $\mathsf{pw} = \mathsf{pw}' \wedge (s^*, T^*) \neq (s, T)$, do the same as above, except that we now find $\mathcal{A}$ *or* P''s $H_{\mathsf{tag}}(\mathsf{pw}, K^*)$ query resulting in τ^*.

The difference between the new Game 2 and the old one is that we now have two separate cases:

1. $\mathsf{pw} \neq \mathsf{pw}' \vee ((s^*, T^*) = (s, T) \wedge c^* \neq c)$;
2. $\mathsf{pw} = \mathsf{pw}' \wedge (s^*, T^*) \neq (s, T)$.

These two cases combined yield $\neg(\mathsf{pw} = \mathsf{pw}' \wedge (s^*, T^*) = (s, T) \wedge c^* = c)$, which is identical to the case in the old Game 2 (i.e., they cover all situations other than the one addressed in Game 1). In case (1) we proceed as in the old Game 2, whereas in case (2) we cannot due to the "attack" in Figure 15; instead, in case (2) we need to include P''s H_{tag} query as if it were $\mathcal{A}$'s.

Claim 7. *Suppose the CBIND property holds for the KEM scheme. In case (1) above, the probability that P' queries $H_{\mathsf{tag}}(\mathsf{pw}, K)$ is negligible.*

Proof. The proof of this claim is essentially part of the proof of Claim 2. P' makes a single H_{tag} query, on (pw', K'). If $\mathsf{pw} \neq \mathsf{pw}'$, then of course P''s query is not $H_{\mathsf{tag}}(\mathsf{pw}, K)$. Now assume $\mathsf{pw} = \mathsf{pw}'$. This implies $(s^*, T^*) = (s, T) \wedge c^* \neq c$ by the condition of the claim. Then the argument in the proof of Claim 2 still goes through, i.e., we can construct a reduction $\mathcal{R}_{\text{CBIND}}$ such that $\Pr[P' \text{ queries } H_{\mathsf{tag}}(\mathsf{pw}, K)] \leq \mathbf{Adv}^{\text{CBIND}}_{\mathcal{R}_{\text{CBIND}}}$. □

The remaining argument is essentially identical to that in the old Game 2.

Games 3–16 are essentially unchanged from the corresponding games in Section 5.3.

New Game I: Now that $(\mathsf{pw}')^*$ — the password guess extracted from (s^*, T^*) — is defined (in Game 15), we can finally add (part of) what's in step 2 of Figure 16. Concretely, we change the game as follows:

In the case where $(s^*, T^*) \neq (s, T) \wedge (\mathsf{pw}')^* = \mathsf{pw} = \mathsf{pw}' \wedge \tau^* = \tau'$, there is a record $\langle \mathsf{pw}, \star, sk \rangle$, and $\mathsf{Decap}(sk, c^*) = K'$, set $SK := SK'$. If $(\mathsf{pw}')^*$ is undefined, treat it as if $(\mathsf{pw}')^* \neq \mathsf{pw}'$. (In some sense this is the most crucial new game, as it covers the scenario in Figure 15 — which is why we need to make all these changes to begin with. It also corresponds to step 2(c)(ii) in the simulator in Figure 16. We will deal with step 2(c)(i) in new Game III later.)

Since $\mathsf{pw} = \mathsf{pw}' \wedge (s^*, T^*) \neq (s, T)$, Game 16 finds $\mathcal{A}$ or P''s H_{tag} query resulting in τ^*. P' makes a single query to H_{tag}, on input (pw', K'); and the query result is $\tau' = \tau^*$. Therefore, Game 16 finds this query, and does the following:

1. Check if $\mathsf{pw}' = \mathsf{pw}$, which is indeed the case per the condition of the game.
2. Find the $\langle \mathsf{pw}, \star, sk \rangle$ record.
3. Compute $K := \mathsf{Decap}(sk, c^*)$ and check if $K = K'$, which is again the case per the condition of the game.
4. Set $SK := H_{\mathsf{key}}(K) (= H_{\mathsf{key}}(K') = SK')$.

This entire process in Game 16 eventually results in setting $SK := SK'$, which exactly matches the new game. We have

$$\mathbf{Dist}_{\mathcal{Z}}^{16,\mathrm{newI}} = 0.$$

New Game II: In the case where $\mathsf{pw} = \mathsf{pw}' \wedge (s^*, T^*) \neq (s, T)$ (i.e., case (2) in Game 2), we find $\mathcal{A}$ *or* P''s $H_{\mathsf{tag}}(\mathsf{pw}^*, K^*)$ query resulting in τ^* — except that in new Game I's condition, new Game I already shortcuts it to $SK := SK'$ without trying to find any H_{tag} query. In this game, we do not check P''s H_{tag} query anymore, and only check $\mathcal{A}$'s H_{tag} query.

There are the following cases:

- If $\tau^* \neq \tau'$, then of course P''s H_{tag} query does not result in τ^* (it results in τ'), so we can safely drop P''s H_{tag} query from the check.
- If $\tau^* = \tau'$, but there is no $\langle \mathsf{pw}, \star, sk \rangle$ record, then both new Game I and new Game II outright set $SK := \perp$, regardless of H_{tag} queries by $\mathcal{A}$ or P'. Thus, the two games are identical.
- If $\tau^* = \tau'$, there is a record $\langle \mathsf{pw}, pk, sk \rangle$, but $(\mathsf{pw}')^* \neq \mathsf{pw}' \vee \mathsf{Decap}(sk, c^*) \neq K'$, then P''s $H_{\mathsf{tag}}(\mathsf{pw}', K')$ query results in τ^*, and we need to argue that this query can be dropped from the check.
  - If $\mathcal{A}$ queries $H_{\mathsf{tag}}(\mathsf{pw}', K')$, then new Game II still finds this query, so new Game I and new Game II are identical.
  - If $\mathcal{A}$ does not query $H_{\mathsf{tag}}(\mathsf{pw}', K')$, then new Game II sets $SK := \perp$. We argue that new Game I also sets $SK := \perp$ except with negligible probability. New Game I finds P''s $H_{\mathsf{tag}}(\mathsf{pw}', K')$ query, and sets $SK := \perp$ if $\mathsf{Decap}(sk, c^*) \neq K'$. The condition of this case says $(\mathsf{pw}')^* \neq \mathsf{pw}' \vee \mathsf{Decap}(sk, c^*) \neq K'$, so we only need to upper-bound $\Pr[\mathsf{Decap}(sk, c^*) = K']$ given $(\mathsf{pw}')^* \neq \mathsf{pw}'$. By Claim 6, (pk, sk) are independent of pk', [30] which suggests a reduction $\mathcal{R}_{\mathrm{KBIND}}$ to the KBIND property of the KEM scheme:

[30] Claim 6 additionally requires $c^* = c$, but its proof does not use this condition.

Reduction $\mathcal{R}_{\text{KBIND}}$:

Sample $j \leftarrow [q_2 + 1]$ as a guess that the j-th H_2 query (by either $\mathcal{A}$ or P') is $H_2(\mathsf{pw}, T^*)$.
On input (pk, pk', c, K'),

1. Invoke $\mathcal{Z}$. On $(\mathsf{NewSession}, sid, P, P', \mathsf{pw}, \text{“initiator”})$ from $\mathcal{Z}$, sample $s \leftarrow \{0,1\}^{3n}, T \leftarrow \mathbb{G}$ and send (s, T) to $\mathcal{A}$.
   When $\mathcal{A}$ makes an $H_2(\mathsf{pw}, T) =: t$ query, compute

   $$r := s \oplus t, \quad R := T/pk.$$

   (pk is part of $\mathcal{R}_{\text{KBIND}}$'s input.) If $H_1(\mathsf{pw}, r)$ has already been defined and it is not R, abort. Otherwise program $H_1(\mathsf{pw}, r) := R$.
2. On the j-th $H_2(\mathsf{pw}, T') =: t'$ query (by either $\mathcal{A}$ or $\mathcal{R}_{\text{KBIND}}$ itself on behalf of P' — see step 3 below), compute

   $$r' := s \oplus t', \quad R' := T/pk'.$$

   (pk' is part of $\mathcal{R}_{\text{KBIND}}$'s input.) If $H_1(\mathsf{pw}, r')$ has already been defined and it is not R', abort. Otherwise program $H_1(\mathsf{pw}, r') := R'$.
3. On $(\mathsf{NewSession}, sid, P', P, \mathsf{pw}', \text{“respondent”})$ from $\mathcal{Z}$ and (s^*, T^*) from $\mathcal{A}$, if $\neg(\mathsf{pw} = \mathsf{pw}' \wedge (s^*, T^*) \neq (s, T))$, abort. Otherwise
   (a) If $\mathcal{A}$ has not queried $H_2(\mathsf{pw}, T^*)$, make this query on behalf of P'.
   (b) Compute $SK' := H_{\mathsf{key}}(K')$ and $\tau' := H_{\mathsf{tag}}(\mathsf{pw}, K')$. Output SK' to $\mathcal{Z}$, and send (c, τ') to $\mathcal{A}$. (c and K' are part of $\mathcal{R}_{\text{KBIND}}$'s input.)
4. On (c^*, τ^*) from $\mathcal{A}$, if $\tau^* \neq \tau'$, abort. Otherwise output c^*.

By an argument similar to that under Game 5 in Section 4.2, $\mathcal{R}_{\text{KBIND}}$'s aborting probability in steps 1 or 2 is upper-bounded by $q_1(q_2 + 1)/2^{3n}$. Assuming $\mathcal{R}_{\text{KBIND}}$ does not abort in steps 1 or 2, $\mathsf{Decap}(sk, c^*) = K'$, and $\mathcal{R}_{\text{KBIND}}$'s guess of index j is correct, the only difference between new Game I/II and $\mathcal{R}_{\text{KBIND}}$'s simulation (until $\mathcal{A}$ sends c^*) is that $\mathcal{R}_{\text{KBIND}}$ uses its input pk to program $H_1(\mathsf{pw}, r)$, uses its input pk' to program $H_1(\mathsf{pw}', r')$, and then uses its inputs $(c, K') \leftarrow \mathsf{Encap}(pk')$ as the P'-to-P message and while computing SK' and τ'. Since (c, K') are also computed as $\mathsf{Encap}(pk')$ in new Game I/II, $\mathcal{R}_{\text{KBIND}}$ simulates new Game I/II perfectly. Furthermore, $\mathcal{R}_{\text{KBIND}}$ wins the KBIND game. We have

$$\Pr[\mathsf{Decap}(sk, c^*) = K'] \leq (q_2 + 1)\mathbf{Adv}^{\text{KBIND}}_{\mathcal{R}_{\text{KBIND}}} + \frac{q_1(q_2 + 1)}{2^{3n}}.$$

The remaining case is that $(\mathsf{pw}')^* = \mathsf{pw}' \wedge \tau^* = \tau'$, there is a $\langle \mathsf{pw}, \star, sk \rangle$ record, and $\mathsf{Decap}(sk, c^*) = K'$. The condition of the game additionally requires

$\mathsf{pw} = \mathsf{pw}' \wedge (s^*, T^*) \neq (s, T)$. Combined, we get "$(s^*, T^*) \neq (s, T) \wedge (\mathsf{pw}')^* = \mathsf{pw} = \mathsf{pw}' \wedge \tau^* = \tau'$, there is a record $\langle \mathsf{pw}, \star, sk \rangle$, and $\mathsf{Decap}(sk, c^*) = K'$". This is exactly the condition in new Game I, in which case we do not make any change in new Game II. Overall, we conclude that

$$\mathbf{Dist}_{\mathcal{Z}}^{\text{newI,newII}} \leq (q_2 + 1)\mathbf{Adv}_{\mathcal{R}_{\text{KBIND}}}^{\text{KBIND}} + \frac{q_1(q_2+1)}{2^{3n}}.$$

Game 17 is essentially unchanged from the corresponding game in Section 5.3.

New Game III: In the case where $(s^*, T^*) \neq (s, T) \wedge (\mathsf{pw}')^* = \mathsf{pw}' \neq \mathsf{pw} \wedge \tau^* = \tau'$, there is a record $\langle \mathsf{pw}', \star, sk \rangle$, and $\mathsf{Decap}(sk, c^*) = K'$, set $SK := \perp$. (This corresponds to step 2(c)(i) in Figure 16.)

In this case Game 17 tries to find $\mathcal{A}$'s (unique) $H_{\mathsf{tag}}(\mathsf{pw}^*, K^*)$ query resulting in τ^*, and checks if $\mathsf{pw}^* = \mathsf{pw}$ (see Game 13). But $\tau^* = \tau' = H_{\mathsf{tag}}(\mathsf{pw}', K')$, so Game 17 sets $SK := \perp$ because it either fails to find $\mathcal{A}$'s $H_{\mathsf{tag}}(\mathsf{pw}', K')$ query, or it finds such a query but then sets $SK := \perp$ because $\mathsf{pw}^* = \mathsf{pw}' \neq \mathsf{pw}$. Either way, Game 17 matches new Game III which outright sets $SK := \perp$. We have

$$\mathbf{Dist}_{\mathcal{Z}}^{17,\text{newIII}} = 0.$$

Summary. We have made the following essential changes to the games:

1. In the case where $\mathsf{pw} = \mathsf{pw}' \wedge (s^*, T^*) \neq (s, T)$ (case (2) in Game 2), Game 2 tries to find P''s H_{tag} query resulting in τ^* (in addition to $\mathcal{A}$'s query).

2. In the case where $(s^*, T^*) \neq (s, T) \wedge (\mathsf{pw}')^* = \mathsf{pw} = \mathsf{pw}' \wedge \tau^* = \tau'$, there is a record $\langle \mathsf{pw}, \star, sk \rangle$, and $\mathsf{Decap}(sk, c^*) = K'$, new Game I outright sets $SK := SK'$ (without trying to find any H_{tag} query by $\mathcal{A}$ or P').

3. In the case where $\mathsf{pw} = \mathsf{pw}' \wedge (s^*, T^*) \neq (s, T)$, new Game II drops the check on P''s H_{tag} query introduced in Game 2 (except in new Game I's condition, where all checks on H_{tag} queries have already been dropped).

4. In the case where $(s^*, T^*) \neq (s, T) \wedge (\mathsf{pw}')^* = \mathsf{pw}' \neq \mathsf{pw} \wedge \tau^* = \tau'$, there is a record $\langle \mathsf{pw}', \star, sk \rangle$, and $\mathsf{Decap}(sk, c^*) = K'$, new Game III outright sets $SK := \perp$.

Items (1) and (3) introduce the check on P''s H_{tag} query and then drop it, so there is no net change there. To see the net change, we only need to combine items (2) and (4), which say that in the case where $(s^*, T^*) \neq (s, T) \wedge (\mathsf{pw}')^* = \mathsf{pw}' \wedge \tau^* = \tau'$, there is a record $\langle \mathsf{pw}', \star, sk \rangle$, and $\mathsf{Decap}(sk, c^*) = K'$,

- If $(\mathsf{pw}')^* = \mathsf{pw}$, then we set $SK := H_{\mathsf{key}}(K')$ [item (2)].
- If $(\mathsf{pw}')^* \neq \mathsf{pw}$, then we set $SK := \perp$ [item (4)].

This exactly matches step 2 of the simulator in Figure 16, so new Game III is identical to the ideal world. This completes the proof.

The "hash pw and c" version. If we define τ as $H_{\mathsf{tag}}(\mathsf{pw}, c, K)$, then Claim 7 unconditionally holds, and the protocol can be proven secure under the UNPRE and KBIND properties of the KEM scheme (without the CBIND property).

B.3 The "Hash pk" Version

We now move on to the variant where pw, but not pk, is further removed from H_{tag} inputs, i.e., τ is defined as $H_{\mathsf{tag}}(pk, K)$. This would best be compared with the protocol (and its security proof) in Section 5.4, where τ is defined as $H_{\mathsf{tag}}(pk, s, T, K)$; as well as the protocol (and its security proof) in Appendix B.1, where τ is defined as $H_{\mathsf{tag}}(\mathsf{pw}, pk, K)$. Again there is a new scenario that requires additional care from the simulator, so we start from the changes to the simulator.

B.3.1 Changes to the Simulator

Similar to Appendix B.2, once we remove pk from H_{tag}, the adversary $\mathcal{A}$ can force the two parties' session keys to be the same without performing an "actual" attack. However, this time the two parties' passwords may or may not be the same (and $\mathcal{A}$ needs to know both of them), and $\mathcal{A}$ should forward both c and τ' from P' without modification. We show this scenario in Figure 17.

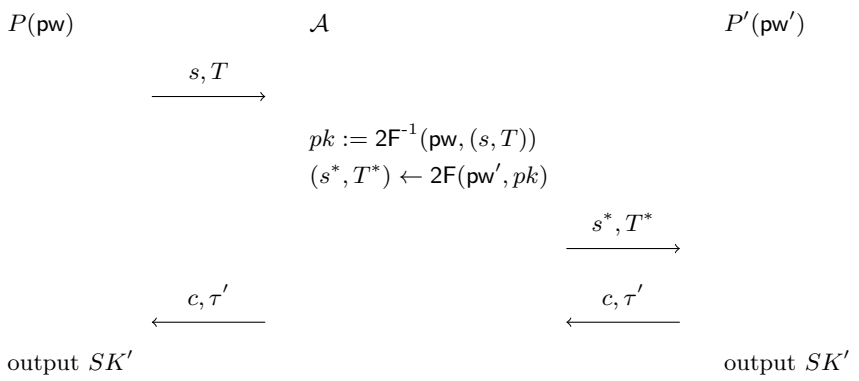

Figure 17: New "attack" on the "hash pk" version

In this "attack", the two parties' passwords pw and pw' may or may not be the same. (The special case where $\mathsf{pw} = \mathsf{pw}'$ is the "attack" in Figure 13.) Assuming $\mathcal{A}$ knows both, it can decrypt P's message to recover pk, and then re-encrypt pk under pw'; this ensures that P' will recover the same pk as on the P side, and use it to compute c, K', SK', τ'. But then if $\mathcal{A}$ forwards (c, τ') without any modification, P will decapsulate c to the same K' (because it uses the same pk), and its session key will be the same as SK' on the P' side. (This is not an issue if pw is included in H_{tag}, since forwarding $\tau' = H_{\mathsf{tag}}(\mathsf{pw}', \dots) \neq H_{\mathsf{tag}}(\mathsf{pw}, \dots)$ to P would make P's check fail unless there is a collision in H_{tag}.)

This new situation should be simulated as follows: When $\mathcal{A}$ decrypts (s, T) under pw, the simulator $\mathcal{S}$ records $\langle \mathsf{pw}, pk, sk \rangle$ (although there might be a large number of such decryption attempts, and $\mathcal{S}$ does not know which is the "right" one at this point). Then when $\mathcal{A}$ sends (s^*, T^*) to P', $\mathcal{S}$ extracts pw', *which also gives a* pk'. Later, when $\mathcal{A}$ forwards (c, τ'), $\mathcal{S}$ knows that $\mathcal{A}$ is performing the aforementioned attack, and checks which $\langle \mathsf{pw}, pk, sk \rangle$ record matches pk'. In this way $\mathcal{S}$ can extract the password guess pw on the P side.

We show the new simulator in Figure 18. (Only the steps after receiving (c^*, τ^*) are included.)

P's session key

On (c^*, τ^*) from $\mathcal{F}$, simulate P's output as follows:

1. If $(s^*, T^*) = (s, T) \wedge (c^*, \tau^*) = (c, \tau')$, send $(\mathsf{NewKey}, sid, P, 0^n)$ to $\mathcal{F}$. // $\mathcal{A}$ is passive
2. If $(s^*, T^*) \neq (s, T) \wedge (c^*, \tau^*) = (c, \tau')$, and there has been a $(\mathsf{TestPwd}, sid, P', \mathsf{pw}')$ command to $\mathcal{F}$ resulting in "correct guess" (i.e., $\mathcal{S}$ previously entered step 2(b)(ii) in Figure 7), there must be a pk' and an SK' defined at that time.
   (a) Retrieve record $\langle \mathsf{pw}^*, pk', \star \rangle$. If there is more than one such record, abort. If there is no such record, go to step 3.
   (b) Send $(\mathsf{TestPwd}, sid, P, \mathsf{pw}^*)$ to $\mathcal{F}$.
      i. If $\mathcal{F}$ replies with "wrong guess", send $(\mathsf{NewKey}, sid, P, 0^n)$ to $\mathcal{F}$.
      ii. If $\mathcal{F}$ replies with "correct guess", send $(\mathsf{NewKey}, sid, P, SK')$ to $\mathcal{F}$.
3. Otherwise find $\mathcal{A}$'s $H_{\mathsf{tag}}(pk^*, K^*)$ query resulting in τ^*.
   (a) If there is no such query or more than one such query, send $(\mathsf{TestPwd}, sid, P, \perp)$ and then $(\mathsf{NewKey}, sid, P, 0^n)$ to $\mathcal{F}$.
   (b) If there is exactly one such query, retrieve record $\langle \mathsf{pw}^*, \star, sk^* \rangle$. If there is more than one such record, abort. If there is no such record, send $(\mathsf{TestPwd}, sid, P, \perp)$ and then $(\mathsf{NewKey}, sid, P, 0^n)$ to $\mathcal{F}$.
   (c) Compute $K := \mathsf{Decap}(sk^*, c^*)$. If $K^* \neq K$, send $(\mathsf{TestPwd}, sid, P', \perp)$ and then $(\mathsf{NewKey}, sid, P, 0^n)$ to $\mathcal{F}$.
   (d) Send $(\mathsf{TestPwd}, sid, P, \mathsf{pw}^*)$ to $\mathcal{F}$.
      i. If $\mathcal{F}$ replies with "wrong guess", send $(\mathsf{NewKey}, sid, P, 0^n)$ to $\mathcal{F}$.
      ii. If $\mathcal{F}$ replies with "correct guess", compute $SK := H_{\mathsf{key}}(K)$ and send $(\mathsf{NewKey}, sid, P, SK)$ to $\mathcal{F}$.

Figure 18: UC simulator $\mathcal{S}$ for the "hash pk" variant of OEKE. Only the paragraph that needs essential change from Figure 11 (shown in blue) is included

B.3.2 Changes in Game Hops

Again, we only describe Game 2 and the four new games (there are four new games rather than three, as we need to additionally consider the aborting condition in step 2(a) of Figure 18).

Game 2: In the case where $(\mathsf{pw} \neq \mathsf{pw}' \wedge (s^*, T^*) = (s, T)) \vee c^* \neq c$, find $\mathcal{A}$'s $H_{\mathsf{tag}}(pk, K^*)$ query resulting in τ^*. If there is none or more than one, set $SK := \perp$. If there is a unique such query, do the following:

- If $K^* = K$, set $SK := H_{\mathsf{key}}(K)$ (i.e., there is no change from Game 1).
- If $K^* \neq K$, set $SK := \perp$.

Furthermore, in the case where $(s^*, T^*) \neq (s, T) \wedge c^* = c$, do the same as above, except that we now find $\mathcal{A}$ *or* P''s $H_{\mathsf{tag}}(pk, K^*)$ query resulting in τ^*.

The difference between the new Game 2 and the old one is that we now have two separate cases:

1. $(\mathsf{pw} \neq \mathsf{pw}' \wedge (s^*, T^*) = (s, T)) \vee c^* \neq c$;
2. $(s^*, T^*) \neq (s, T) \wedge c^* = c$.

These two cases combined yield $\neg(\mathsf{pw} = \mathsf{pw}' \wedge (s^*, T^*) = (s, T) \wedge c^* = c)$, which is identical to the case in the old Game 2 (i.e., they cover all situations other than the one addressed in Game 1). In case (1) we proceed as in the old Game 2, whereas in case (2) we cannot due to the "attack" in Figure 17; instead, in case (2) we need to include P''s H_{tag} query as if it were $\mathcal{A}$'s.

Claim 8. *Suppose the CBIND property holds for the KEM scheme. In case (1) above, the probability that P' queries $H_{\mathsf{tag}}(pk, K)$ is negligible.*

Proof. P' makes a single H_{tag} query, on (pk', K'). There are the following cases:

- If $\mathsf{pw} \neq \mathsf{pw}' \wedge (s^*, T^*) = (s, T)$, then the argument in the proof of Claim 3 still goes through, so $\Pr[pk = pk'] \leq 1/|\mathbb{G}|$.
- If $c^* \neq c$, we claim that $pk = pk' \wedge K = K'$ happens with negligible probability. Suppose $pk = pk'$. Then we can reduce the event $K = K'$ to the CBIND property of the KEM scheme, via a reduction $\mathbf{Adv}^{\mathrm{CBIND}}_{\mathcal{R}_{\mathrm{CBIND}}}$ similar to that in the proof of Claim 2. We have $\Pr[pk = pk' \wedge K = K'] \leq \mathbf{Adv}^{\mathrm{CBIND}}_{\mathcal{R}_{\mathrm{CBIND}}}$.

If $pk \neq pk'$ or $K \neq K'$ then of course P''s query is not $H_{\mathsf{tag}}(pk, K)$. Overall, we have

$$\Pr[P' \text{ queries } H_{\mathsf{tag}}(pk, K)] \leq \mathbf{Adv}^{\mathrm{CBIND}}_{\mathcal{R}_{\mathrm{CBIND}}} + \frac{1}{|\mathbb{G}|},$$

which yields the claim. □

The remaining argument is essentially identical to that in the old Game 2.

Games 3–16 are essentially unchanged from the corresponding games in Section 5.3.

New game I: Now that $(\mathsf{pw}')^*$ — the password guess extracted from (s^*, T^*) — is defined (in Game 15), we can finally add (part of) what's in step 2 of Figure 18. In this game we first add the aborting condition. Concretely, in the case where $(s^*, T^*) \neq (s, T) \wedge (\mathsf{pw}')^* = \mathsf{pw}' \wedge (c^*, \tau^*) = (c, \tau')$, there must be a pk' defined on the P' side. If there is more than one record $\langle \mathsf{pw}^*, pk', \star \rangle$, abort. If $(\mathsf{pw}')^*$ is undefined, treat it as if $(\mathsf{pw}')^* \neq \mathsf{pw}'$. (This corresponds to step 2(a) in the simulator in Figure 18.)

In each $\langle \mathsf{pw}^*, pk^*, \star \rangle$ record, $(pk^*, \star) \leftarrow \mathsf{Gen}$ is freshly generated, so the probability that one of these pk^* happens to be equal to pk' is upper-bounded by $q_2 \cdot p_{\text{guess}}$ (recall that there is a $\langle \mathsf{pw}^*, pk^*, sk^* \rangle$ record for each $H_2(\mathsf{pw}^*, T)$ query). We have

$$\mathbf{Dist}_{\mathcal{Z}}^{16,\text{newI}} \leq q_2 \cdot p_{\text{guess}}.$$

New game II: Again in the case where $(s^*, T^*) \neq (s, T) \wedge (\mathsf{pw}')^* = \mathsf{pw}' \wedge (c^*, \tau^*) = (c, \tau')$, if there is a unique record $\langle \mathsf{pw}^*, pk', \star \rangle$, and $\mathsf{pw}^* = \mathsf{pw}$, set $SK := SK'$. (In some sense this is the most crucial new game, as it covers the scenario in Figure 17 — which is why we need to make all these changes to begin with. It also corresponds to step 2(b)(ii) in the simulator in Figure 18. We will deal with step 2(b)(i) later.)

Since $(s^*, T^*) \neq (s, T) \wedge c^* = c$, Game 16 finds $\mathcal{A}$ or P''s H_{tag} query resulting in τ^*. P' makes a single query to H_{tag}, on input (pk', K'); and the query result is $\tau' = \tau^*$. Therefore, Game 16 finds this query, and does the following (see the description of Game 13 in Section 5.4.2):

1. Find the (unique) record $\langle \mathsf{pw}^*, pk', sk' \rangle$.
2. Check if $\mathsf{pw}^* = \mathsf{pw}$, which is indeed the case per the condition of the game.
3. Compute $K := \mathsf{Decap}(sk', c^*)$ and check if $K = K'$. Steps 1 and 2 above mean that there is a (unique) $\langle \mathsf{pw}, pk', sk' \rangle$ record. But then

$$(c, K') \leftarrow \mathsf{Encap}(pk') \text{ on the } P' \text{ side},$$
$$c^* = c,$$
$$K = \mathsf{Decap}(sk', c^*) \text{ on the } P \text{ side},$$

   and (pk', sk') are an honestly generated KEM public/secret key pair (again, because there is a $\langle \mathsf{pw}, pk', sk' \rangle$ record), so by the correctness of the KEM scheme, the $K' \stackrel{?}{=} K$ check passes except with probability $\varepsilon_{\text{correctness}}$.
4. Set $SK := H_{\mathsf{key}}(K) (= H_{\mathsf{key}}(K') = SK')$.

This entire process in Game 16 eventually results in setting $SK := SK'$ except with probability $\varepsilon_{\text{correctness}}$, which exactly matches the new game. We have

$$\mathbf{Dist}_{\mathcal{Z}}^{\text{newI,newII}} \leq \varepsilon_{\text{correctness}}.$$

New game III: In the case where $(s^*, T^*) \neq (s, T) \wedge c^* = c$ (i.e., case (2) in Game 2), we find $\mathcal{A}$ *or* P''s $H_{\mathsf{tag}}(pk^*, K^*)$ query resulting in τ^* — except that in new Game II's condition, new Game II already shortcuts it to $SK := SK'$ without trying to find any H_{tag} query. In this game, we do not check P''s H_{tag} query anymore, and only check $\mathcal{A}$'s H_{tag} query.

There are the following cases:

- If $\tau^* \neq \tau'$, then of course P''s H_{tag} query does not result in τ^* (it results in τ'), so we can safely drop P''s H_{tag} query.
- If $\tau^* = \tau'$, but there is more than one $\langle \mathsf{pw}^*, pk', \star \rangle$ record, then both new Game II and new Game III abort, regardless of H_{tag} queries by $\mathcal{A}$ or P'. Similarly, if $\tau^* = \tau'$, but there is no $\langle \mathsf{pw}^*, pk', \star \rangle$ record, then both new Game II and new Game III outright set $SK := \bot$. Thus, the two games are identical.
- If $\tau^* = \tau'$ and there is a unique $\langle \mathsf{pw}^*, pk', \star \rangle$ record, but $(\mathsf{pw}')^* \neq \mathsf{pw}' \vee \mathsf{pw}^* \neq \mathsf{pw}$, then P''s $H_{\mathsf{tag}}(pk', K')$ query results in τ^*, and we need to argue that this query can be dropped from the check.
  - If $\mathcal{A}$ queries $H_{\mathsf{tag}}(pk', K')$, then new Game II still finds this query, so new Game II and new Game III are identical.
  - If $\mathcal{A}$ does not query $H_{\mathsf{tag}}(pk', K')$, then new Game III sets $SK := \bot$. We argue that new Game II also sets $SK := \bot$ except with negligible probability. New Game II finds P''s $H_{\mathsf{tag}}(pk', K')$ query, then finds the (unique) $\langle \mathsf{pw}^*, pk', sk' \rangle$ record, and sets $SK := \bot$ unless $\mathsf{pw}^* = \mathsf{pw} \wedge \mathsf{Decap}(sk', c^*) = K'$. The condition of this case says $(\mathsf{pw}')^* \neq \mathsf{pw}' \vee \mathsf{pw}^* \neq \mathsf{pw}$, so we only need to upper-bound $\Pr[\mathsf{pw}^* = \mathsf{pw}]$ given $(\mathsf{pw}')^* \neq \mathsf{pw}'$. Recall that pw^* is the candidate password whose record $\langle \mathsf{pw}^*, pk', \star \rangle$ matches pk'. The record for pw is $\langle \mathsf{pw}, pk, \star \rangle$, so $\mathsf{pw}^* = \mathsf{pw}$ is equivalent to $pk = pk'$, whose probability is $1/|\mathbb{G}|$ by Claim 6.

The remaining case is that $(\mathsf{pw}')^* = \mathsf{pw}' \wedge \tau^* = \tau'$, there is a unique $\langle \mathsf{pw}^*, pk', \star \rangle$ record, and $\mathsf{pw}^* = \mathsf{pw}$. The condition of the game additionally requires $(s^*, T^*) \neq (s, T) \wedge c^* = c$. Combined, we get "$(s^*, T^*) \neq (s, T) \wedge (\mathsf{pw}')^* = \mathsf{pw}' \wedge (c^*, \tau^*) = (c, \tau')$, there is a unique record $\langle \mathsf{pw}^*, pk', \star \rangle$, and $\mathsf{pw}^* = \mathsf{pw}$". This is exactly the condition in new Game II, in which case we do not make any change in new Game III. Overall, we conclude that

$$\mathbf{Dist}_{\mathcal{Z}}^{\text{newII,newIII}} \leq \frac{1}{|\mathbb{G}|}.$$

Game 17 is essentially unchanged from the corresponding game in Section 5.3.

New Game IV: In the case where $(s^*, T^*) \neq (s, T) \wedge (c^*, \tau^*) = (c, \tau')$, there is a unique record $\langle \mathsf{pw}^*, pk', \star \rangle$, and $\mathsf{pw}^* \neq \mathsf{pw}$, set $SK := \bot$. (This corresponds to step 2(c)(i) in Figure 18.)

In this case Game 17 first tries to find $\mathcal{A}$'s (unique) $H_{\mathsf{tag}}(pk^*, K^*)$ query resulting in τ^*. Since $\tau^* = \tau' = H_{\mathsf{tag}}(pk', K')$, if $\mathcal{A}$ does not query $H_{\mathsf{tag}}(pk', K')$, then Game 17 sets $SK := \perp$. Otherwise Game 17 finds $\mathcal{A}$'s $H_{\mathsf{tag}}(pk', K')$ query, then tries to find the (unique) $\langle \mathsf{pw}^*, pk', \star \rangle$ record, and checks if $\mathsf{pw}^* = \mathsf{pw}$ (see Game 13). Since $\mathsf{pw}^* \neq \mathsf{pw}$, the check never passes, so Game 17 again sets $SK := \perp$. Either way, Game 17 matches new Game IV which outright sets $SK := \perp$. We have

$$\mathbf{Dist}_{\mathcal{Z}}^{17,\text{newIV}} = 0.$$

Summary. We have made the following essential changes to the games:

1. In the case where $(s^*, T^*) \neq (s, T) \wedge c^* = c$, Game 2 tries to find P''s H_{tag} query resulting in τ^* (in addition to $\mathcal{A}$'s query).
2. In the case where $(s^*, T^*) \neq (s, T) \wedge (\mathsf{pw}')^* = \mathsf{pw}' \wedge (c^*, \tau^*) = (c, \tau')$, and there is more than one record $\langle \mathsf{pw}^*, pk', \star \rangle$, new Game I aborts.
3. In the case where $(s^*, T^*) \neq (s, T) \wedge (c^*, \tau^*) = (c, \tau')$, there is a unique record $\langle \mathsf{pw}^*, pk', \star \rangle$, and $\mathsf{pw}^* = \mathsf{pw}$, new Game II outright sets $SK := SK'$ (without trying to find any H_{tag} query by $\mathcal{A}$ or P').
4. In the case where $(s^*, T^*) \neq (s, T) \wedge c^* = c$, new Game III drops the check on P''s H_{tag} query introduced in Game 2 (except in new Game II's condition, where all checks on H_{tag} queries have already been dropped).
5. In the case where $(s^*, T^*) \neq (s, T) \wedge (c^*, \tau^*) = (c, \tau')$, there is a unique record $\langle \mathsf{pw}^*, pk', \star \rangle$, and $\mathsf{pw}^* \neq \mathsf{pw}$, new Game IV outright sets $SK := \perp$.

Items (1) and (4) introduce the check on P''s H_{tag} query and then drop it, so there is no net change there. To see the net change, we only need to combine items (2), (3) and (5), which say that in the case where $(s^*, T^*) \neq (s, T) \wedge (c^*, \tau^*) = (c, \tau')$ and there is a record $\langle \mathsf{pw}^*, pk', \star \rangle$,

- If the record is not unique, then the game aborts [item (2)].
- If the record is unique, and $\mathsf{pw}^* = \mathsf{pw}$, then we set $SK := SK'$ [item (3)].
- If the record is unique, and $\mathsf{pw}^* \neq \mathsf{pw}$, then we set $SK := \perp$ [item (5)].

This exactly matches step 2 of the simulator in Figure 18, so new Game IV is identical to the ideal world. This completes the proof.

The "hash pk and c" version. If we define τ as $H_{\mathsf{tag}}(pk, c, K)$, then Claim 8 unconditionally holds, and the protocol can be proven secure without the CBIND property of the KEM scheme (i.e., under the same assumptions as in the "hash everything" version).

B.4 The "Hash Nothing But K" Version

The last variant we consider is that both pw and pk (as well as c) are removed from H_{tag} inputs, i.e., τ is defined as the "bare" $H_{\mathsf{tag}}(K)$. There is an even more general "attack" against this version that the simulator must take into consideration, which we start from.

B.4.1 Changes to the Simulator

We show the new situation in Figure 19, using the specific OEKE with Diffie–Hellman KEM.

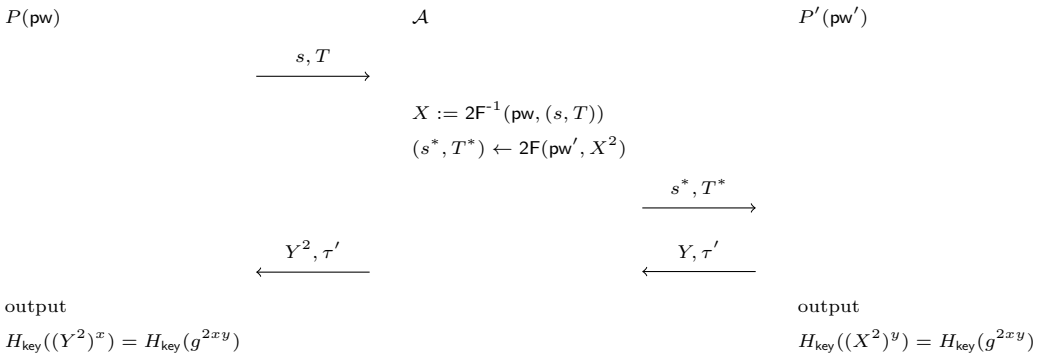

Figure 19: New "attack" on the "hash nothing but K" version, using OEKE based on Diffie–Hellman KEM

In this "attack", the two parties' passwords may or may not be the same. Assuming $\mathcal{A}$ knows both, it can decrypt P's message to recover $X = g^x$, and then re-encrypt X^2 under pw'; this causes P' to output $SK' = H_{\mathsf{key}}(g^{2xy})$ and send $(Y = g^y, \tau' = H_{\mathsf{tag}}(g^{2xy}))$. Then $\mathcal{A}$ sends (Y^2, τ') to P, causing P's check to pass and P will also output $H_{\mathsf{key}}(g^{2xy})$. This is a generalization of both "attacks" in Appendix B.2 (where $\mathcal{A}$ can change pk but not pw during re-encryption) and Appendix B.3 (where $\mathcal{A}$ can change pw but not pk during re-encryption); here, since neither pw nor pk is included in H_{tag} inputs, $\mathcal{A}$ can change *both* of them.

The simulation strategy in this scenario is essentially a combination of the simulators in Appendices B.2 and B.3. When $\mathcal{A}$ decrypts (s, T) under pw, $\mathcal{S}$ records $\langle \mathsf{pw}, pk, sk \rangle$ (although there might be a large number of such decryption attempts, and $\mathcal{S}$ does not know which is the "right" one at this point). $\mathcal{S}$ also computes K' while simulating the P' side. Later, when $\mathcal{A}$ sends (c^*, τ'), $\mathcal{S}$ can tell if $\mathcal{A}$ is performing the aforementioned attack by checking if there exists a record $\langle \mathsf{pw}, pk, sk \rangle$ that satisfies $\mathsf{Decap}(sk, c^*) = K'$ (using the specific example in Figure 19: When $\mathcal{A}$ sends (Y^2, τ'), $\mathcal{S}$ finds the record $\langle \mathsf{pw}, X, x \rangle$ that satisfies $(Y^2)^x = K'$, where $K' = (X^2)^y$ was computed while encapsulating X^2

on (s^*, T^*) from $\mathcal{A}$). In this way $\mathcal{S}$ can extract the password guess pw on the P side.

We show the new simulator in Figure 20. (Only the steps after receiving (c^*, τ^*) are included.)

P's session key

On (c^*, τ^*) from $\mathcal{F}$, simulate P's output as follows:

1. If $(s^*, T^*) = (s, T) \wedge (c^*, \tau^*) = (c, \tau')$, send $(\mathsf{NewKey}, sid, P, 0^n)$ to $\mathcal{F}$. // $\mathcal{A}$ is passive
2. If $(s^*, T^*) \neq (s, T) \wedge \tau^* = \tau'$, and there has been a $(\mathsf{TestPwd}, sid, P, \mathsf{pw}')$ command to $\mathcal{F}$ resulting in "correct guess" (i.e., $\mathcal{S}$ previously entered step 2(b)(ii) in Figure 7), there must be a K' and an SK' defined at that time.
   (a) Retrieve all records in the form of $\langle \mathsf{pw}^*, \star, sk^* \rangle$. For each such record, compute $K := \mathsf{Decap}(sk^*, c^*)$ and check if $K = K'$. If there is more than one such record, abort. If there is no such record, go to step 3.
   (b) If there is exactly one record $\langle \mathsf{pw}^*, \star, sk^* \rangle$ with $K := \mathsf{Decap}(sk^*, c^*) = K'$, send $(\mathsf{TestPwd}, sid, P, \mathsf{pw}^*)$ to $\mathcal{F}$.
      i. If $\mathcal{F}$ replies with "wrong guess", send $(\mathsf{NewKey}, sid, P, 0^n)$ to $\mathcal{F}$.
      ii. If $\mathcal{F}$ replies with "correct guess", send $(\mathsf{NewKey}, sid, P, SK')$ to $\mathcal{F}$.
3. Otherwise find $\mathcal{A}$'s $H_{\mathsf{tag}}(K^*)$ query resulting in τ^*.
   (a) If there is no such query or more than one such query, send $(\mathsf{TestPwd}, sid, P, \perp)$ and then $(\mathsf{NewKey}, sid, P, 0^n)$ to $\mathcal{F}$.
   (b) If there is exactly one such query, retrieve all records in the form of $\langle \mathsf{pw}^*, \star, sk^* \rangle$. For each such record, compute $K := \mathsf{Decap}(sk^*, c^*)$ and check if $K^* = K$. If there is more than one such record, abort. If there is no such record, send $(\mathsf{TestPwd}, sid, P, \perp)$ and then $(\mathsf{NewKey}, sid, P, 0^n)$ to $\mathcal{F}$.
   (c) If there is exactly one record $\langle \mathsf{pw}^*, \star, sk^* \rangle$ with $K := \mathsf{Decap}(sk^*, c^*) = K^*$, send $(\mathsf{TestPwd}, sid, P, \mathsf{pw}^*)$ to $\mathcal{F}$.
      i. If $\mathcal{F}$ replies with "wrong guess", send $(\mathsf{NewKey}, sid, P, 0^n)$ to $\mathcal{F}$.
      ii. If $\mathcal{F}$ replies with "correct guess", compute $SK := H_{\mathsf{key}}(K)$ and send $(\mathsf{NewKey}, sid, P, SK)$ to $\mathcal{F}$.

Figure 20: UC simulator $\mathcal{S}$ for the "hash pk" variant of OEKE. Only the paragraph that needs essential change from Figure 12 (shown in blue) is included

B.4.2 Changes in Game Hops

Once more, we only describe Game 2 and the four new games.

Game 2: In the case where $(s^*, T^*) = (s, T) \wedge (\mathsf{pw} \neq \mathsf{pw}' \vee c^* \neq c)$, find $\mathcal{A}$'s $H_{\mathsf{tag}}(K^*)$ query resulting in τ^*. If there is none or more than one, set $SK := \perp$. If there is a unique such query, do the following:

- If $K^* = K$, set $SK := H_{\mathsf{key}}(K)$ (i.e., there is no change from Game 1).
- If $K^* \neq K$, set $SK := \perp$.

Furthermore, in the case where $(s^*, T^*) \neq (s, T)$, do the same as above, except that we now find $\mathcal{A}$ *or* P''s $H_{\mathsf{tag}}(K^*)$ query resulting in τ^*.

The difference between the new Game 2 and the old one is that we now have two separate cases:

1. $(s^*, T^*) = (s, T) \wedge (\mathsf{pw} \neq \mathsf{pw}' \vee c^* \neq c)$;
2. $(s^*, T^*) \neq (s, T)$.

These two cases combined yield $\neg(\mathsf{pw} = \mathsf{pw}' \wedge (s^*, T^*) = (s, T) \wedge c^* = c)$, which is identical to the case in the old Game 2 (i.e., they cover all situations other than the one addressed in Game 1). In case (1) we proceed as in the old Game 2, whereas in case (2) we cannot due to the "attack" in Figure 19; instead, in case (2) we need to include P''s H_{tag} query as if it were $\mathcal{A}$'s.

Claim 9. *Suppose the CBIND and KBIND properties hold for the KEM scheme. In case (1) above, the probability that P' queries $H_{\mathsf{tag}}(K)$ is negligible.*

Proof. This has been proven in Claim 4 (most of the proof of Claim 4 assumes $(s^*, T^*) = (s, T)$, which is also satisfied here). □

Games 3–16 are essentially unchanged from the corresponding games in Section 5.5.

New game I: Now that $(\mathsf{pw}')^*$ — the password guess extracted from (s^*, T^*) — is defined (in Game 15), we can finally add (part of) what's in step 2 of Figure 20. In this game we first add the aborting condition. Concretely, in the case where $(s^*, T^*) \neq (s, T) \wedge (\mathsf{pw}')^* = \mathsf{pw}' \wedge \tau^* = \tau'$, there must be a K' defined on the P' side. Retrieve all records in the form of $\langle \mathsf{pw}^*, \star, sk^* \rangle$. If there is more than one such record such that $\mathsf{Decap}(sk^*, c^*) = K'$, abort. If $(\mathsf{pw}')^*$ is undefined, treat it as if $(\mathsf{pw}')^* \neq \mathsf{pw}'$. (This corresponds to step 2(a) in the simulator in Figure 20.)

Note that in each such record, $(\star, sk^*) \leftarrow \mathsf{Gen}$ is freshly generated, so a straightforward reduction to the CR property of the KEM scheme, $\mathcal{R}_{\mathrm{CR2}}$, shows that the probability that the same c^* decapsulates to the same K' under two freshly generated sk^* is upper-bounded by $\binom{q_2}{2} \cdot \mathbf{Adv}^{\mathrm{CR}}_{\mathcal{R}_{\mathrm{CR2}}}$ (cf. Game 13 in Section 5.4.2). We have

$$\mathbf{Dist}^{16,\mathrm{newI}}_{\mathcal{Z}} \leq \binom{q_2}{2} \cdot \mathbf{Adv}^{\mathrm{CR}}_{\mathcal{R}_{\mathrm{CR2}}}.$$

New game II: Again in the case where $(s^*, T^*) \neq (s, T) \wedge (\mathsf{pw}')^* = \mathsf{pw}' \wedge \tau^* = \tau'$, if there is a unique record $\langle \mathsf{pw}^*, \star, sk^* \rangle$ such that $\mathsf{Decap}(sk^*, c^*) = K'$, and $\mathsf{pw}^* = \mathsf{pw}$, set $SK := SK'$. (In some sense this is the most crucial new game, as it covers the scenario in Figure 19 — which is why we need to make all these changes to begin with. It also corresponds to step 2(b)(ii) in the simulator in Figure 20. We will deal with step 2(b)(i) later.)

Since $(s^*, T^*) \neq (s, T)$, Game 16 finds $\mathcal{A}$ or P''s H_{tag} query resulting in τ^*. P' makes a single query to H_{tag}, on input K'; and the query result is $\tau' = \tau^*$. Therefore, Game 16 finds this query, and does the following:

1. Retrieve all records in the form of $\langle \mathsf{pw}^*, \star, sk^* \rangle$, and check if there is such a (unique) record such that $K := \mathsf{Decap}(sk^*, c^*) = K'$. This is indeed the case per the condition of the game.
2. Check if $\mathsf{pw}^* = \mathsf{pw}$, which is again the case per the condition of the game.
3. Set $SK := H_{\mathsf{key}}(K) (= H_{\mathsf{key}}(K') = SK')$.

This entire process in Game 16 eventually results in setting $SK := SK'$, which exactly matches the new game. We have

$$\mathbf{Dist}_{\mathcal{Z}}^{\text{newI,newII}} = 0.$$

New game III: In the case where $(s^*, T^*) \neq (s, T)$ (i.e., case (2) in Game 2), we find $\mathcal{A}$ *or* P''s $H_{\mathsf{tag}}(K^*)$ query resulting in τ^* — except that in new Game II's condition, new Game II already shortcuts it to $SK := SK'$ without trying to find any H_{tag} query. In this game, we do not check P''s H_{tag} query anymore, and only check $\mathcal{A}$'s H_{tag} query.

There are the following cases:

- If $\tau^* \neq \tau'$, then of course P''s H_{tag} query does not result in τ^* (it results in τ'), so we can safely drop P''s H_{tag} query.
- If $\tau^* = \tau'$, but there is more than one $\langle \mathsf{pw}^*, \star, sk^* \rangle$ record such that $\mathsf{Decap}(sk^*, c^*) = K'$, then both new Game II and new Game III abort, regardless of H_{tag} queries by $\mathcal{A}$ or P'. Similarly, if $\tau^* = \tau'$, but there is no $\langle \mathsf{pw}^*, \star, sk^* \rangle$ record such that $\mathsf{Decap}(sk^*, c^*) = K'$, then both new Game II and new Game III outright set $SK := \perp$. Thus, the two games are identical.
- If $\tau^* = \tau'$ and there is a unique $\langle \mathsf{pw}^*, \star, sk^* \rangle$ record, but $(\mathsf{pw}')^* \neq \mathsf{pw}' \vee \mathsf{pw}^* \neq \mathsf{pw}$, then P''s $H_{\mathsf{tag}}(K')$ query results in τ^*, and we need to argue that this query can be dropped from the check.
  - If $\mathcal{A}$ queries $H_{\mathsf{tag}}(K')$, then new Game II still finds this query, so new Game II and new Game III are identical.
  - If $\mathcal{A}$ does not query $H_{\mathsf{tag}}(K')$, then new Game III sets $SK := \perp$. We argue that new Game II also sets $SK := \perp$ except with negligible probability. New Game II finds P''s $H_{\mathsf{tag}}(K')$ query, then finds the

(unique) $\langle \mathsf{pw}^*, \star, sk^* \rangle$ record such that $\mathsf{Decap}(sk^*, c^*) = K'$, and sets $SK := \perp$ unless $\mathsf{pw}^* = \mathsf{pw}$. The condition of this case says $(\mathsf{pw}')^* \neq \mathsf{pw}' \vee \mathsf{pw}^* \neq \mathsf{pw}$, so we only need to upper-bound $\Pr[\mathsf{pw}^* = \mathsf{pw}]$ given $(\mathsf{pw}')^* \neq \mathsf{pw}'$. By the definition of pw^*, $\mathsf{pw}^* = \mathsf{pw}$ is equivalent to there is a $\langle \mathsf{pw}, pk, sk \rangle$ record such that $\mathsf{Decap}(sk, c^*) = K'$. pk and pk' are independent of each other by Claim 6, so a reduction to the KBIND property of the KEM scheme similar to that in new Game II of Appendix B.2.2, $\mathcal{R}_{\text{KBIND2}}$, shows that $\Pr[\mathsf{pw}^* = \mathsf{pw}] \leq (q_2 + 1)\mathbf{Adv}^{\text{KBIND}}_{\mathcal{R}_{\text{KBIND2}}}$.

The remaining case is that $(\mathsf{pw}')^* = \mathsf{pw}' \wedge \tau^* = \tau'$, there is a unique $\langle \mathsf{pw}^*, \star, sk^* \rangle$ record such that $\mathsf{Decap}(sk^*, c^*) = K'$, and $\mathsf{pw}^* = \mathsf{pw}$. The condition of the game additionally requires $(s^*, T^*) \neq (s, T)$. Combined, we get "$(s^*, T^*) \neq (s, T) \wedge (\mathsf{pw}')^* = \mathsf{pw}' \wedge \tau^* = \tau'$, there is a unique $\langle \mathsf{pw}^*, \star, sk^* \rangle$ record such that $\mathsf{Decap}(sk^*, c^*) = K'$, and $\mathsf{pw}^* = \mathsf{pw}$". This is exactly the condition in new Game II, in which case we do not make any change in new Game III. Overall, we conclude that

$$\mathbf{Dist}^{\text{newII,newIII}}_{\mathcal{Z}} \leq (q_2 + 1)\mathbf{Adv}^{\text{KBIND}}_{\mathcal{R}_{\text{KBIND2}}}.$$

Game 17 is essentially unchanged from the corresponding game in Section 5.3.

New Game IV: In the case where $(s^*, T^*) \neq (s, T) \wedge \tau^* = \tau'$, if there is a unique record $\langle \mathsf{pw}^*, \star, sk^* \rangle$ such that $\mathsf{Decap}(sk^*, c^*) = K'$, but $\mathsf{pw}^* \neq \mathsf{pw}$, set $SK := \perp$. (This corresponds to step 2(c)(i) in Figure 20.)

In this case Game 17 first tries to find $\mathcal{A}$'s (unique) $H_{\mathsf{tag}}(K^*)$ query resulting in τ^*. Since $\tau^* = \tau' = H_{\mathsf{tag}}(K')$, if $\mathcal{A}$ does not query $H_{\mathsf{tag}}(K')$, then Game 17 sets $SK := \perp$. Otherwise Game 17 finds $\mathcal{A}$'s $H_{\mathsf{tag}}(K')$ query, then tries to find the (unique) $\langle \mathsf{pw}^*, \star, sk^* \rangle$ record such that $\mathsf{Decap}(sk^*, c^*) = K'$, and checks if $\mathsf{pw}^* = \mathsf{pw}$ (see Game 13). Since $\mathsf{pw}^* \neq \mathsf{pw}$, the check never passes, so Game 17 again sets $SK := \perp$. Either way, Game 17 matches new Game IV which outright sets $SK := \perp$. We have

$$\mathbf{Dist}^{17,\text{newIV}}_{\mathcal{Z}} = 0.$$

Summary. We have made the following essential changes to the games:

1. In the case where $(s^*, T^*) \neq (s, T)$, Game 2 tries to find P''s H_{tag} query resulting in τ^* (in addition to $\mathcal{A}$'s query).

2. In the case where $(s^*, T^*) \neq (s, T) \wedge (\mathsf{pw}')^* = \mathsf{pw}' \wedge \tau^* = \tau'$, and there is more than one record $\langle \mathsf{pw}^*, \star, sk^* \rangle$ such that $\mathsf{Decap}(sk^*, c^*) = K'$, new Game I aborts.

3. In the case where $(s^*, T^*) \neq (s, T) \wedge \tau^* = \tau'$, there is a unique record $\langle \mathsf{pw}^*, \star, sk^* \rangle$ such that $\mathsf{Decap}(sk^*, c^*) = K'$, and $\mathsf{pw}^* = \mathsf{pw}$, new Game II outright sets $SK := SK'$ (without trying to find any H_{tag} query by $\mathcal{A}$ or P').

4. In the case where $(s^*, T^*) \neq (s, T)$, new Game III drops the check on P''s H_{tag} query introduced in Game 2 (except in new Game II's condition, where all checks on H_{tag} queries have already been dropped).

5. In the case where $(s^*, T^*) \neq (s, T) \wedge \tau^* = \tau'$, there is a unique record $\langle \mathsf{pw}^*, \star, sk^* \rangle$ such that $\mathsf{Decap}(sk^*, c^*) = K'$, and $\mathsf{pw}^* \neq \mathsf{pw}$, new Game IV outright sets $SK := \perp$.

Items (1) and (4) introduce the check on P''s H_{tag} query and then drop it, so there is no net change there. To see the net change, we only need to combine items (2), (3) and (5), which say that in the case where $(s^*, T^*) \neq (s, T) \wedge \tau^* = \tau'$ and there is a record $\langle \mathsf{pw}^*, \star, sk^* \rangle$ such that $\mathsf{Decap}(sk^*, c^*) = K'$,

- If the record is not unique, then the game aborts [item (2)].
- If the record is unique, and $\mathsf{pw}^* = \mathsf{pw}$, then we set $SK := SK'$ [item (3)].
- If the record is unique, and $\mathsf{pw}^* \neq \mathsf{pw}$, then we set $SK := \perp$ [item (5)].

This exactly matches step 2 of the simulator in Figure 20, so new Game IV is identical to the ideal world. This completes the proof.

The "hash c" version. If we define τ as $H_{\mathsf{tag}}(c, K)$, then the reduction to CBIND in Claim 9 is not needed, and the protocol can be proven secure under the KBIND and CR properties of the KEM scheme.

C Comparison with [JRX25]

We believe our security proof for the "hash nothing but K" version in Appendix B.4 is fundamentally similar to the OEKE security proof in [JRX25]; however, the structure and presentation of their proof are drastically different from ours. Below we give a high-level comparison with [JRX25], although a game-by-game comparison — like what we did with [ABJ25] in Section 4.2 — seems impossible.

Modular structure of [JRX25]. While our proof is a monolith, the proof in [JRX25] is split into three levels:

- [JRX25, Section 4] gives an ideal functionality $\mathcal{F}_{\text{POPF}}$ for the 2-round Feistel cipher. In fact the 2F construction does not appear to UC-realize $\mathcal{F}_{\text{POPF}}$ (or any reasonable UC functionality); to circumvent this, [JRX25] only proves that 2F "almost" UC-realizes $\mathcal{F}_{\text{POPF}}$, a relaxation of UC which places some restrictions on the upper-level protocol composed with 2F. We refer to [JRX25, Section 4] for more details.
- [JRX25, Section 5.1] proves that the P-to-P' message, using $\mathcal{F}_{\text{POPF}}$, UC-realizes a "first-round" functionality $\mathcal{F}_{\text{EKE-1r}}$. The reason why they do this is that the P-to-P' message in EKE and OEKE is exactly identical,

so having this common first-round functionality essentially allows a large chunk of argument to be reused across the two security proofs. However, if we only care about OEKE then this abstraction is unnecessary.

- Finally, [JRX25, Section 5.3] proves that OEKE-2F (the "hash nothing but K" version), using $\mathcal{F}_{\text{EKE-1r}}$, UC-realizes the standard PAKE functionality $\mathcal{F}_{\text{PAKE}}$. (They do not consider the PAKE functionality with initiator-side explicit authentication.)

If we collect and combine all proofs in the three sections of [JRX25] using the composition theorem, that should yield a monolith security proof for OEKE. However, there would be so much work that we might as well perform a monolith proof from scratch — as what we did in this work.

KEM scheme: hashed v. unhashed. Another difficulty when comparing our security proof with the one in [JRX25] is that they put the ROs H_{key} and H_{tag} inside the KEM scheme.[31] In our description of OEKE, we use the KEM scheme to derive the KEM key K, and then use H_{key} and H_{tag} to derive the PAKE session key SK and the authenticator τ', respectively. In contrast, [JRX25] splits the KEM key into $SK||\tau'$, without applying an RO — which means that the ROs are implicitly "baked into" the KEM scheme.

In more detail, let $(\mathsf{Gen}, \mathsf{Encap}, \mathsf{Decap})$ be a KEM scheme as used in our description of OEKE. We can construct another KEM scheme $(\mathsf{Gen}, \mathsf{Encap}', \mathsf{Decap}')$ as follows:

- $\mathsf{Encap}'(pk)$: Run $(c, K) \leftarrow \mathsf{Encap}(pk)$, compute $SK := H_{\mathsf{key}}(K)$ and $\tau' := H_{\mathsf{tag}}(K)$, and output $(c, SK||\tau')$.
- $\mathsf{Decap}'(sk, c)$: Run $K := \mathsf{Decap}(sk, c)$, compute $SK := H_{\mathsf{key}}(K)$ and $\tau' := H_{\mathsf{tag}}(K)$, and output $SK||\tau'$.

Then on the PAKE level, the OEKE scheme can use the "raw" KEM key as SK and τ', without mentioning H_{key} or H_{tag}.

Outputting the "raw" KEM key also means that [JRX25] needs stronger KEM security properties than ours. The OEKE security result in [JRX25] requires three KEM security properties: first pseudorandomness, pseudorandom non-malleability, and collision resistance. First pseudorandomness is the same as what we call UNI-PK. Pseudorandom non-malleability roughly corresponds to IND-1CCA$, whereas we only need OW-1PCA and ANO-1PCA. (It shouldn't be surprising that they need IND and we only need OW: they use the "raw" KEM key as the PAKE session key, whereas we use an RO of the KEM key — so we only need the KEM key to be unpredictable given the KEM ciphertext.) Collision resistance is stronger than our CR, in that it does not require the adversary to output the KEM key (among other things).

[31] In fact, [JRX25] uses the syntax of (unauthenticated) key exchange protocols, rather than KEM.

Remark 12. [ABJ25, Section 1] correctly observes that [JRX25] requires stronger KEM security properties, but does not specify *why*. There are, in fact, two orthogonal reasons:

1. [JRX25] puts the ROs inside the KEM scheme, so it needs "indistinguishability" rather than "unprecitability" type of properties;
2. [JRX25] only hashes K in H_{tag}, whereas [ABJ25] hashes everything.

(1) explains why [JRX25] needs pseudorandom non-malleability, and (2) explains why [JRX25] needs collision resistance.

D Deferred Proofs from Section 2

D.1 Proof of Lemma 1

We need two helper lemmas first.

Lemma 4. *Suppose the UNI-PK and OW properties hold for a KEM scheme. Then for any PPT adversary $\mathcal{A}$, the probability that $\mathcal{A}$ wins the following game is negligible:*

$pk \leftarrow \mathcal{PK}$
$(c, K) \leftarrow \mathsf{Encap}(pk)$
$K^* \leftarrow \mathcal{A}(pk, c)$
$\mathcal{A}$ wins if $K^* = K$

(Below we call this property UNI-OW.)

Proof. This is straightforward so we only provide a sketch. We claim that $\mathcal{A}$'s winning probabilities in the following two games are negligibly close:

UNI-OW game:
$pk \leftarrow \mathcal{PK}$
$(c, K) \leftarrow \mathsf{Encap}(pk)$
$K^* \leftarrow \mathcal{A}(pk, c)$
$\mathcal{A}$ wins if $K^* = K$

OW game:
$(pk, \star) \leftarrow \mathsf{Gen}$
$(c, K) \leftarrow \mathsf{Encap}(pk)$
$K^* \leftarrow \mathcal{A}(pk, c)$
$\mathcal{A}$ wins if $K^* = K$

This is due to a reduction $\mathcal{R}_{\text{UNI-PK}}$ to the UNI-PK property of the KEM scheme: $\mathcal{R}_{\text{UNI-PK}}$, on input pk, can run either the UNI-OW game or the OW-game — depending on whether its challenge bit $b = 1$ (i.e., pk is uniformly random) or $b = 0$ (i.e., pk is honestly generated) — and output 1 if $\mathcal{A}$ wins and 0 otherwise. The difference between $\Pr[\mathcal{A}$ wins the UNI-OW game$]$ and $\Pr[\mathcal{A}$ wins the OW game$]$ is equal to $\mathcal{R}_{\text{UNI-PK}}$'s advantage, which must be negligible. Since the OW property holds for the KEM scheme, $\Pr[\mathcal{A}$ wins the OW game$]$ is also negligible. So $\Pr[\mathcal{A}$ wins the UNI-OW game$]$ must be negligible. □

Lemma 5. *Suppose the UNI-PK and ANO properties hold for a KEM scheme. Then for any PPT adversary $\mathcal{A}$, $\Pr[b^* = 1]$ in the following two games are negligibly close:*

challenge bit $b = 0$:	challenge bit $b = 1$:
$pk \leftarrow \mathcal{PK}$	$pk, pk' \leftarrow \mathcal{PK}$
$(c, \star) \leftarrow \mathsf{Encap}(pk)$	$(c, \star) \leftarrow \mathsf{Encap}(pk')$
$b^* \leftarrow \mathcal{A}(pk, c)$	$b^* \leftarrow \mathcal{A}(pk, c)$

(Below we call this property UNI-ANO.)

Proof. The proof goes by a sequence of games, and we again only provide a sketch. The following two games

challenge bit $b = 0$:	challenge bit $b = 0$ in the ANO game:
$pk \leftarrow \mathcal{PK}$	$(pk, \star) \leftarrow \mathsf{Gen}$
$(c, \star) \leftarrow \mathsf{Encap}(pk)$	$(c, \star) \leftarrow \mathsf{Encap}(pk)$
$b^* \leftarrow \mathcal{A}(pk, c)$	$b^* \leftarrow \mathcal{A}(pk, c)$

are indistinguishable due to the UNI-PK property of the KEM scheme (the reduction is very similar to that in the proof of Lemma 4, except that here the reduction copies $\mathcal{A}$'s output bit instead of outputting whether $\mathcal{A}$ wins or not); the following two games

challenge bit $b = 0$ in the ANO game:	challenge bit $b = 1$ in the ANO game:
$(pk, \star) \leftarrow \mathsf{Gen}$	$(pk, \star) \leftarrow \mathsf{Gen}$
$(c, \star) \leftarrow \mathsf{Encap}(pk)$	$(pk', \star) \leftarrow \mathsf{Gen}$
$b^* \leftarrow \mathcal{A}(pk, c)$	$(c, \star) \leftarrow \mathsf{Encap}(pk')$
	$b^* \leftarrow \mathcal{A}(pk, c)$

are indistinguishable because this is exactly what the ANO property of the KEM scheme says; finally, the following two games

challenge bit $b = 1$ in the ANO game:	challenge bit $b = 1$:
$(pk, \star) \leftarrow \mathsf{Gen}$	$pk, pk' \leftarrow \mathcal{PK}$
$(pk', \star) \leftarrow \mathsf{Gen}$	$(c, \star) \leftarrow \mathsf{Encap}(pk')$
$(c, \star) \leftarrow \mathsf{Encap}(pk')$	$b^* \leftarrow \mathcal{A}(pk, c)$
$b^* \leftarrow \mathcal{A}(pk, c)$	

are indistinguishable again due to the UNI-PK property of the KEM scheme. □

Proof of Lemma 1. The proof goes by a sequence of games, and we again only provide a sketch. Suppose $\mathcal{A}$ makes q_1, q_2 queries to ROs H_1, H_2, respectively. We start from the game where the challenge bit $b = 0$. To begin with, we claim that $\Pr[b^* = 1]$ does not change if we simplify the game so that $\mathcal{A}$ cannot choose

its index i:

challenge bit $b = 0$:
$pk_1, \ldots, pk_q \leftarrow \mathcal{PK}$
$i \leftarrow \mathcal{A}(pk_1, \ldots, pk_q)$
$(c, K) \leftarrow \mathsf{Encap}(pk_i)$
$h_1 := H_1(K); h_2 := H_2(K)$
$b^* \leftarrow \mathcal{A}(c, h_1, h_2)$

simplified $b = 0$ game:
$pk \leftarrow \mathcal{PK}$
$(c, \star) \leftarrow \mathsf{Encap}(pk)$
$h_1 := H_1(K); h_2 := H_2(K)$
$b^* \leftarrow \mathcal{A}(pk, c, h_1, h_2)$

Note that no matter which index i $\mathcal{A}$ chooses, once i is fixed $\mathcal{A}$'s view is identical to its view in the simplified game (plus redundant public keys pk_{i^*} for $i^* \neq i$, which are not used anywhere in the game and thus do not affect $\Pr[b^* = 1]$). Therefore, letting

$$p_{i^*} = \Pr[b^* = 1 \mid i = i^* \text{ in the original game}],$$
$$p = \Pr[b^* = 1 \text{ in the simplified game}],$$

we have that for any $i^* \in [q]$,

$$p_{i^*} = p.$$

But then

$$\begin{aligned}\Pr[b^* = 1 \text{ in the original game}] &= \sum_{i^*=1}^{q} p_{i^*} \cdot \Pr[i = i^*] \\ &= \sum_{i^*=1}^{q} p \cdot \Pr[i = i^*] \\ &= p \cdot \sum_{i^*=1}^{q} \Pr[i = i^*] = p,\end{aligned}$$

so $\Pr[b^* = 1]$ in the original game and the simplified game are equal. (Note that we do not need a hybrid argument here.)

Next, we sample h_1, h_2 at random rather than set them to the real RO outputs:

simplified $b = 0$ game:
$pk \leftarrow \mathcal{PK}$
$(c, \star) \leftarrow \mathsf{Encap}(pk)$
$h_1 := H_1(K); h_2 := H_2(K)$
$b^* \leftarrow \mathcal{A}(pk, c, h_1, h_2)$

intermediate game:
$pk \leftarrow \mathcal{PK}$
$(c, K) \leftarrow \mathsf{Encap}(pk)$
$h_1 \leftarrow \{0,1\}^n; h_2 \leftarrow \{0,1\}^{2n}$
$b^* \leftarrow \mathcal{A}(pk, c, h_1, h_2)$

The two games are identical unless $\mathcal{A}$ queries either $H_1(K)$ or $H_2(K)$, which happens with negligible probability due to a reduction $\mathcal{R}_{\text{UNI-OW}}$ to the UNI-OW property of the KEM scheme (Lemma 4). $\mathcal{R}_{\text{UNI-OW}}$, on input (pk, c), samples $i \leftarrow [q_1 + q_2]$ as a guess that $\mathcal{A}$'s i-th H_1 or H_2 query is on K. Then $\mathcal{R}_{\text{UNI-OW}}$ samples $pk \leftarrow \mathcal{PK}, h_1 \leftarrow \{0,1\}^n, h_2 \leftarrow \{0,1\}^{2n}$ and invokes

$\mathcal{A}$ on input (pk, c, h_1, h_2). On $\mathcal{A}$'s i-th H_1 or H_2 query (say the query input is K^*), $\mathcal{R}_{\text{UNI-OW}}$ outputs K^*. Assuming $\mathcal{A}$ queries either $H_1(K)$ or $H_2(K)$, and $\mathcal{R}_{\text{UNI-OW}}$'s guess of index i is correct, $\mathcal{R}_{\text{UNI-OW}}$ simulates the two games perfectly until the "bad event" happens and wins the UNI-OW game. So the above two games are indistinguishable (with a $q_1 + q_2$ reduction loss).

After that, we change the generation of c from using pk_i to using a fresh pk':

intermediate game:	simplified $b = 1$ game:
$pk \leftarrow \mathcal{PK}$	$pk, pk' \leftarrow \mathcal{PK}$
$(c, \star) \leftarrow \mathsf{Encap}(pk)$	$(c, \star) \leftarrow \mathsf{Encap}(pk')$
$h_1 \leftarrow \{0,1\}^n; h_2 \leftarrow \{0,1\}^{2n}$	$h_1 \leftarrow \{0,1\}^n; h_2 \leftarrow \{0,1\}^{2n}$
$b^* \leftarrow \mathcal{A}(pk, c, h_1, h_2)$	$b^* \leftarrow \mathcal{A}(pk, c, h_1, h_2)$

The two games are indistinguishable, due to the UNI-ANO property of the KEM scheme (Lemma 5). (The only difference is that here $\mathcal{A}$ additionally sees h_1, h_2, which are not used anywhere else in the game.)

Finally, we bring back $\mathcal{A}$'s ability to choose an index i and arrive at the game where the challenge bit $b = 1$:

simplified $b = 1$ game:	challenge bit $b = 1$:
$pk, pk' \leftarrow \mathcal{PK}$	$pk', pk_1, \ldots, pk_q \leftarrow \mathcal{PK}$
$(c, \star) \leftarrow \mathsf{Encap}(pk')$	$i \leftarrow \mathcal{A}(pk_1, \ldots, pk_q)$
$h_1 \leftarrow \{0,1\}^n; h_2 \leftarrow \{0,1\}^{2n}$	$(c, \star) \leftarrow \mathsf{Encap}(pk')$
$b^* \leftarrow \mathcal{A}(pk, c, h_1, h_2)$	$h_1 \leftarrow \{0,1\}^n; h_2 \leftarrow \{0,1\}^{2n}$
	$b^* \leftarrow \mathcal{A}(c, h_1, h_2)$

An argument similar to the first game hop shows that $\Pr[b^* = 1]$ in the two games above are equal, completing the proof. □

D.2 Proof of Lemma 2

Proof. We only provide a sketch. First,

$$\Pr\left[\, K = K' \;:\; \begin{array}{c} (\star, sk) \leftarrow \mathsf{Gen} \\ pk' \leftarrow \mathcal{PK} \\ (c, K') \leftarrow \mathsf{Encap}(pk') \\ K := \mathsf{Decap}(sk, c) \end{array} \right]$$

and

$$\Pr\left[\, K = K' \;:\; \begin{array}{c} (\star, sk) \leftarrow \mathsf{Gen} \\ (pk', \star) \leftarrow \mathsf{Gen} \\ (c, K') \leftarrow \mathsf{Encap}(pk') \\ K := \mathsf{Decap}(sk, c) \end{array} \right] \qquad (\dagger)$$

are negligibly close due to a reduction $\mathcal{R}_{\text{UNI-PK}}$ to the UNI-PK property of the KEM scheme: $\mathcal{R}_{\text{UNI-PK}}$, on input pk', computes $(pk, sk) \leftarrow \mathsf{Gen}$, $(c, K') \leftarrow \mathsf{Encap}(pk')$, and $K := \mathsf{Decap}(sk, c)$, and outputs 1 if $K = K'$ and 0 otherwise.

Next, we show (†) is negligible. Consider the following adversary $\mathcal{A}$ in the OW game for the KEM scheme: $\mathcal{A}$, on input (pk', c), computes $(pk, sk) \leftarrow \mathsf{Gen}$ and $K := \mathsf{Decap}(sk, c)$, and outputs K. $\mathcal{A}$ wins the game if and only if the equation in (†) holds, so

$$(\dagger) = \mathbf{Adv}^{\mathrm{OW}}_{\mathcal{A}},$$

which is negligible. (Note that $\mathcal{A}$ does not use its input pk', so a weaker OW property, where the adversary is only given c, suffices.) □

D.3 Proof of Lemma 3

Proof. The proof is similar to that of [AMO+26, Theorem 2]. For any PPT adversary $\mathcal{A}$ in the UNPRE game, we construct a reduction $\mathcal{R}_{\mathrm{CR}}$ to the CR property of the KEM scheme. $\mathcal{R}_{\mathrm{CR}}$ ignores its input (pk, pk') and invokes $\mathcal{A}$. When $\mathcal{A}$ outputs (c^*, K^*), $\mathcal{R}_{\mathrm{CR}}$ also outputs (c^*, K^*).

We have

$$\begin{aligned}
\mathbf{Adv}^{\mathrm{UNPRE}}_{\mathcal{A}} &= \Pr[\mathsf{Decap}(sk, c^*) = K^*], \\
\mathbf{Adv}^{\mathrm{UNPRE}}_{\mathcal{A}} &= \Pr[\mathsf{Decap}(sk', c^*) = K^*], \\
\mathbf{Adv}^{\mathrm{CR}}_{\mathcal{R}_{\mathrm{CR}}} &= \Pr[\mathsf{Decap}(sk, c^*) = K^* \wedge \mathsf{Decap}(sk', c^*) = K^*].
\end{aligned}$$

For any *fixed* pair of c^* and K^*, events $\mathsf{Decap}(sk, c^*) = K^*$ and $\mathsf{Decap}(sk', c^*) = K^*$ are independent of each other: c^* and K^* completely determine a set S of secret keys sk^* such that $\mathsf{Decap}(sk^*, c^*) = K^*$. Therefore, $\mathsf{Decap}(sk, c^*) = K^*$ is equivalent to $sk \in S$, and $\mathsf{Decap}(sk', c^*) = K^*$ is equivalent to $sk' \in S$; since sk and sk' are generated independently, "$sk \in S$" and "$sk' \in S$" are independent of each other.

However, c^* and K^* are in fact random variables dependent on $\mathcal{A}$'s randomness ρ. Therefore, we need to sum over all fixed pairs of c_0 and K_0.[32] We have (let R be $\mathcal{A}$'s randomness space)

$$\begin{aligned}
&\Pr[\mathsf{Decap}(sk, c^*) = K^* \wedge \mathsf{Decap}(sk', c^*) = K^*] \\
&= \sum_{c_0, K_0} \left(\Pr_{\rho \leftarrow R}[\mathcal{A}(\rho) = (c_0, K_0)] \cdot \Pr_{\substack{(\star, sk) \leftarrow \mathsf{Gen} \\ (\star, sk') \leftarrow \mathsf{Gen}}}[\mathsf{Decap}(sk, c_0) = K_0 \wedge \mathsf{Decap}(sk', c_0) = K_0] \right) \\
&= \sum_{c_0, K_0} \left(\Pr_{\rho \leftarrow R}[\mathcal{A}(\rho) = (c_0, K_0)] \cdot \Pr_{(\star, sk) \leftarrow \mathsf{Gen}}[\mathsf{Decap}(sk, c_0) = K_0]^2 \right) \\
&\geq \left(\sum_{c_0, K_0} \Pr_{\rho \leftarrow R}[\mathcal{A}(\rho) = (c_0, K_0)] \cdot \Pr_{(\star, sk) \leftarrow \mathsf{Gen}}[\mathsf{Decap}(sk, c_0) = K_0] \right)^2 \\
&= \Pr[\mathsf{Decap}(sk, c^*) = K^*]^2,
\end{aligned}$$

[32] A prior version of this work said events $\mathsf{Decap}(sk, c^*) = K^*$ and $\mathsf{Decap}(sk', c^*) = K^*$ (for random variables c^* and K^*) are independent of each other, which is incorrect. This can be said only if $\mathcal{A}$ is deterministic. A similar error appears in [JRX25, Section 5.3, Hybrid 2] ("These are just q independent events").

so

$$\mathbf{Adv}_{\mathcal{A}}^{\mathrm{UNPRE}} \leq \sqrt{\mathbf{Adv}_{\mathcal{R}_{\mathrm{CR}}}^{\mathrm{CR}}},$$

which is negligible. □