On the UC-(In)Security of PAKE Protocols Without the Random Oracle Model

Naman Kumar*
Oregon State University

Jiayu Xu†
Oregon State University

Abstract

A Password-Authenticated Key Exchange (PAKE) protocol allows two parties to jointly establish a shared cryptographic key, where the only information shared in advance is a low-entropy password. The first efficient PAKE protocol whose security does not rely on the random oracle model is the one by Katz, Ostrovsky and Yung (KOY, EUROCRYPT 2001). Unfortunately, the KOY protocol has only been proven secure in the game-based setting, and it is unclear whether KOY is secure in the stronger Universal Composability (UC) framework, which is the current security standard for PAKE.

In this work, we present a thorough study of the UC-security of KOY. Our contributions are two-fold:

1. We formally prove that the KOY protocol is *not* UC-secure;
2. We then show that the UC-security of KOY holds in the Algebraic Group Model, under the Decisional Square Diffie–Hellman (DSDH) assumption.

Overall, we characterize the exact conditions under which KOY is UC-secure. Interestingly, the DSDH assumption is stronger than DDH under which KOY can be proven game-based secure, which reveals some subtle gaps between the two PAKE security notions that have never been studied.

1 Introduction

A *Password-Authenticated Key Exchange (PAKE)* protocol allows two parties to jointly establish a cryptographic key, where the only information shared in advance is a low-entropy string called the *password*. Crucially, PAKE protocols must be secure against man-in-the-middle attackers in the *password-only setting*, where no PKI is available. Such protocols have received increasing attention in recent years, and are interesting from both theoretical and practical perspectives: with the standardization process by the IETF in 2019–20 [Cry20], PAKE is beginning to see broad applications in real life; and theoretically speaking, PAKE bootstraps the shared secret from a low-entropy password to a high-entropy key, and its construction involves a number of interesting techniques and has boosted the development of some underlying primitives such as *Smooth-Projective Hash Functions (SPHFs)*.

*Email: kumarnam@oregonstate.edu
†Email: xujiay@oregonstate.edu. Partially supported by NSF award 2453883

Two paradigms of PAKE construction. The vast majority of PAKE protocols in the literature fall into one of two main paradigms. The first one, starting from the Encrypted Key Exchange (EKE) protocol [BM92], generally involves each party sending its message in an unauthenticated key exchange protocol (such as Diffie–Hellman) encrypted under the password. Protocols in this category include SPAKE1 and SPAKE2 [AP05], and some other protocols such as SPEKE [Jab97] are inspired by EKE. These protocols are generally extremely efficient but heavily rely on the Random Oracle Model (ROM); in some cases [BM92, Jab97], it requires an RO hash into a group, which might be hard to implement.

Another paradigm for PAKE construction makes use of SPHFs, and their security can be proven in the standard model where only a CRS is present. (A PAKE in the "plain" model which does not even use a CRS is extremely difficult to construct and inefficient, and none of the existing protocols [GL01, NV04, GJO10] achieves any standard notion of security.) This line of PAKE protocols began with the seminal work of Katz, Ostrovsky and Yung [KOY01] (henceforth KOY), which is one of the most thoroughly studied PAKE protocols: in particular, it inspired a number of follow-up works including generalization [GL03] and simplification [Gen08, ABP15], as well as extension to the two-server setting [KMTG05]. Subsequent works following this paradigm include [JG04, GK10, KV09, KV11]. Aside from not requiring an RO, SPHF-based PAKE protocols also have reasonable efficiency; for example, the computational cost of the KOY protocol is 3 group exponentiations + 3 multi-exponentiations for each party (plus extremely fast cryptographic operations such as hash functions and the signing/verification of a one-time signature scheme), i.e., slightly higher than 3 times the cost of (unauthenticated) Diffie–Hellman.

PAKE security definition: game-based v. UC. Defining the security of PAKE protocols is a delicate task. There are a number of definitions in the literature, which can be divided into two types: game-based [BPR00, AFP05] and Universally Composable (UC) [CHK+05, ABB+20]. In multi-party computation, UC definitions are stronger than game-based definitions in general, as the former guarantees security under *arbitrary composition* with other protocols; in PAKE, the UC definition has the additional advantage of modeling password reuse or correlated passwords across multiple protocol sessions. For these reasons, the UC definition has become the standard for PAKE security; in particular, all protocols that entered the second round of the IETF competition have a UC-security proof [AHH21, ABB+20, JKX18, HL19].

The UC definitions have been proven to imply the (standard) game-based definition ([CHK+05, Appendix A] and [ABB+20, Section 5]; the UC definition in [CHK+05] is the standard one which we use in this work). The converse is not true, as there are a large number of PAKE protocols that are game-based secure but not UC-secure; in fact, the UC-security of *all efficient PAKE protocols* involves one type of caveats or another. One reason is that UC-security implies *perfect forward secrecy* (even if the password is leaked, the sessions already completed should remain secure), which is not covered by the standard game-based security definition. However, there are more subtle gaps where there is no apparent security weakness yet the UC "simulatability" fails. There has been no systematic study of the gaps between game-based and UC-security, but for all "natural" PAKE protocols that are game-based but not UC-secure, the reason seems to fall into one of the following two categories:

1. *RO-based protocols*: UC-security requires a PPT simulator to extract an adversary's password guess from protocol messages, and this "extractability" property is not required in game-based security. Take SPAKE2 as an example: there, both parties' protocol message is a Pedersen commitment to the password $C = g^x \cdot M^{\mathsf{pw}}$ (for random group element M

which is part of the CRS, and random integer x sampled by the protocol party), which information-theoretically hides the password. This means that even a computationally unbounded simulator cannot extract the password guess from the protocol message.
There are two ways to salvage the UC-security of SPAKE2. First, the session key is derived from hashing the Diffie–Hellman key together with the password and the protocol transcript, which allows for password extraction *after the protocol session ends*. This means that SPAKE2 realizes a weaker "lazy-extraction" version of UC PAKE functionality, where a password guess can be submitted by the simulator even after the session ends. This is the approach in [ABB+20]; its deficiency is that it is unclear what type of forward secrecy this security notion provides.[1]
The other possibility is to analyze the (standard) UC-security of SPAKE2 in the *Algebraic Group Model (AGM)* [FKL18]. Recall that in the AGM, the adversary $\mathcal{A}$ is required to "explain" how it computed all group elements it outputs; that is, for any group element Y output by $\mathcal{A}$, $\mathcal{A}$ needs to provide the *algebraic coefficients* $\alpha_1, \ldots, \alpha_n$ such that

$$Y = X_1^{\alpha_1} \cdots X_n^{\alpha_n},$$

where $X_1, \ldots, X_n$ are group elements $\mathcal{A}$ has seen so far. This means that in the AGM, when the adversary sends protocol message C, it must provide an "explanation" from which the password guess can be extracted immediately.[2] The security of SPAKE2 in the AGM is shown in [ABK+21], which also proves that the UC composition theorem still holds in the AGM.

2. *SPHF-based protocols*: These protocols are not UC-secure for a completely different reason. In the game-based security definition, the adversary "wins" the security game once it guesses the correct password, and all security guarantees are lost. By contrast, the UC definition requires the simulator to *continue simulating the protocol even after a successful attack*; in particular, it must compute the session key of the attacked party and send this value to the UC PAKE functionality. In all "natural" SPHF-based PAKE protocols [KOY01, JG04, KV09, KV11], the simulator can extract the password guess but cannot compute the session key afterwards, because the password extraction happens "too late".
To make SPHF-based PAKE protocols UC-secure (without introducing the ROM), [CHK+05] modifies KOY by adding a "pre-commitment" to the password to allow for early extraction, as well as a Non-Interactive Zero-Knowledge (NIZK) proof that the "pre-commitment" indeed commits to the correct password. This adds one round to the existing protocol, and drastically increases the computational cost.[3] The most frustrating point is that the sole purpose of this NIZK proof — which destroys the efficiency of KOY — seems to be allowing the simulation to go through, and it is not even clear what exact attack this failure of UC simulation implies. Indeed, the trick of adding a NIZK proof has been called "mysterious" [Jar22, Section 10.5].

Given the success of the UC-AGM-security analysis of SPAKE2, a natural question arises:

[1] [ABB+20, Section 5.3] shows that "lazy-extraction" UC PAKE implies weak forward secrecy, but it is unclear whether perfect forward secrecy is also implied.

[2] Formally, we require both the UC environment and the adversary to be algebraic; see [ABK+21] for more details.

[3] KOY requires 3 group exponentiations + 3 multi-exponentiations per party; the protocol in [CHK+05] requires around 30 group exponentiations per party. Later, [KV11] introduces another way to achieve UC-security without increasing the number of rounds, but the overhead of computational cost remains high.

Can we prove the UC-security of the original, reasonably efficient SPHF-based PAKE protocols, in the AGM but without the ROM?

Our contributions. In this work, we perform a thorough study of the UC-(in)security of the KOY protocol and present two results. First, we prove that KOY is *not* UC-secure, even in the ROM.[4] The reason — as briefly discussed above — has been explained intuitively [CHK+05, Section 3.3], but we give the first formal proof of this result; in particular, we show that KOY is UC-insecure *even if the simulator is computationally unbounded*, ruling out the possibility of proving the UC-security of KOY even in the weaker "UC with angels" framework.[5] This forces us to use some other idealized models — such as the AGM — while considering the UC-security of KOY.

Second and more importantly, we prove that KOY (without the additional NIZK proof) is UC-secure in the AGM. In particular, we show that the simulator is able to compute the attacked party's session key using the algebraic coefficients provided by the adversary, hence completing its simulation. This completes the picture of the UC-(in)security of KOY and renders it "usable" in the UC setting. Furthermore, our security analysis reveals some interesting nature of the UC-security definition for PAKE in general. In particular, there is a third gap between game-based and UC-security of PAKE which has never been noticed in the literature:

3. Even if the simulator can complete the simulation by computing the attacked party's session key correctly, the environment might use this information to distinguish the *previously simulated protocol messages* from the real messages. (Again, this issue does not emerge in the game-based setting, since the adversary simply "wins" the security game and the game does not need to proceed any further if the password guess is correct; as such, the adversary does not get to see the attacked party's session key.)

Concretely, the game-based security of KOY holds under the DDH assumption, whereas the UC-AGM-security relies on the stronger Decisional Square Diffie–Hellman (DSDH) assumption, which says that given g^x for random integer x, it is hard to distinguish g^{x^2} from random. In fact, we will show an explicit attack that renders KOY UC-insecure (even in the AGM) if DDH holds in the group but DSDH does not — a surprising result that was unexpected by the authors.

ROM v. AGM: why does it matter? Given that the AGM is also an idealized model (like the ROM), one might ask why not simply use ROM-based PAKE protocols that are faster. Our primary answer is that *the ROM and the AGM are very different idealized models in nature, with their respective advantages and disadvantages that are incomparable.* Security proofs in both the ROM and the AGM are heuristic arguments and do not provide iron-clad security guarantees; however, a proof in the AGM seems easier to interpret: people's faith in the ROM relies on the assumption that the hash function "behaves like a random function", whose exact meaning is unclear (for example, it is hard to quantify how "good" SHA-3 is as an RO); by contrast, a proof in the AGM rules out a limited yet well-defined class of adversaries. In other words, our result shows that a successful attack on (the composability of) KOY must rely on some

[4] KOY uses a hash function where only collision-resistance is required; we show the UC-insecurity of KOY even if the hash function is modeled as an RO.

[5] UC with angels [PS04] gives both the real adversary and the simulator access to an oracle (the "angel") that solves some hard problems such as DDH. It has been used in the security analyses of some asymmetric PAKE protocols such as SRP (the first PAKE protocol widely used in practice) [DL24, XZ25].

non-algebraic methods, where the adversary comes up with new group elements via means other than performing group operations on group elements it has seen. This provides more concrete guidance for both implementers and cryptanalysts.[6]

In addition, (as mentioned above) KOY is one of the most well-studied PAKE protocols and represents a major paradigm of PAKE construction, so studying its security level is of significant importance in its own right. Finally, the gaps between game-based and UC-security definitions for PAKE have always been a somewhat fuzzy topic, and as we have seen, the current understanding is incomplete; as a conceptual contribution, we hope to shed some light in this regard using KOY as a case study.

Our analysis of the UC-(in)security of KOY requires careful considerations of how the simulator can(not) be constructed. Below we first describe the KOY protocol, and then provide an informal but detailed overview of our technical arguments. (While KOY is a classic example of SPHF-based PAKE protocols, our results do not explicitly use the concept of SPHF. To ease the burden on readers who are unfamiliar with SPHF, we intentionally avoid SPHF jargon except in some footnotes.)

1.1 Protocol Description

We provide a high-level description of the KOY protocol along with an intuitive explanation of its security below; for a formal UC description, see Fig. 3. Let $\mathbb{G}$ be a cyclic group in which DDH is believed to be hard and define common random string $\mathsf{crs} := (g_1, g_2, h, c, d)$ to be uniformly sampled from $\mathbb{G}^5$. Note that crs can be interpreted to be a (degenerate) public key of the Cramer–Shoup encryption scheme [CS98, CS03] whose corresponding secret key is unknown.

Call the two protocol parties client and server, who share a password pw. The protocol messages are computed as follows:

- client begins by generating a pair of keys $(\mathsf{VK}, \mathsf{SK})$ for a one-time signature scheme and encrypts its password pw with label VK using the Cramer–Shoup encryption scheme with crs as the public key. That is, let H be a collision-resistant hash function and r_1 be the randomness used during encryption; the resulting ciphertext consists of four group elements $\mathsf{ct}_1 := (A, B, C, D) = (g_1^{r_1}, g_2^{r_1}, h^{r_1} \cdot \mathsf{pw}, (cd^{\alpha})^{r_1})$ where $\alpha = H(\mathsf{VK}, A, B, C)$. client then sends $\mathsf{msg}_1 := (\mathsf{VK}, \mathsf{ct}_1)$ to server.

- Upon receiving $(\mathsf{VK}, A, B, C, D)$, server samples its random exponents (x_2, y_2, z_2, w_2), and computes $E := g_1^{x_2} g_2^{y_2} h^{z_2} (cd^{\alpha})^{w_2}$.[7] Furthermore, server encrypts pw with label (msg_1, E) using Cramer–Shoup encryption (similar to how client encrypts pw in msg_1). Let r_2 be the randomness used in encryption; the resulting ciphertext is $\mathsf{ct}_2 := (F, G, I, J) = (g_1^{r_2}, g_2^{r_2}, h^{r_2} \cdot \mathsf{pw}, (cd^{\beta})^{r_2})$ where $\beta = H(\mathsf{VK}, F, G, I)$. server sends $\mathsf{msg}_2 := (E, \mathsf{ct}_2)$ to client.

- Symmetrically, client upon receiving (E, ct_2) samples its own random exponents (x_1, y_1, z_1, w_1) and computes $K := g_1^{x_1} g_2^{y_1} h^{z_1} (cd^{\beta})^{w_1}$. It signs the protocol transcript $\sigma \leftarrow \mathsf{Sign}_{\mathsf{SK}}(\mathsf{msg}_1, \mathsf{msg}_2, K)$, and sends $\mathsf{msg}_3 := (K, \sigma)$ to server.

[6] Some researchers claim that the fundamental goal of provable security is to provide guidance for cryptanalysis [Ber13]. While one might not agree with this conclusion (e.g., we believe that theoretical questions are interesting for their own sake even if there is no real-world application whatsoever), it is true that this is *one of* the main goals of provable security. The AGM is particularly useful in this regard: to break the security of KOY under composition, one should focus on non-algebraic methods (unless they can break the underlying DSDH assumption).

[7] Note that $\alpha = H(\mathsf{VK}, A, B, C)$, so server must wait for client's message before proceeding.

- server verifies the signature σ using VK, and aborts if it is invalid. Otherwise the session key sk is defined as the product of

$$X_1 = E^{r_1} = A^{x_2} B^{y_2} \left(\frac{C}{\mathsf{pw}}\right)^{z_2} D^{w_2}$$

and

$$X_2 = K^{r_2} = F^{x_1} G^{y_1} \left(\frac{I}{\mathsf{pw}}\right)^{z_1} J^{w_1}.$$

client can compute X_1 as E^{r_1} and X_2 as $F^{x_1} G^{y_1} (I/\mathsf{pw})^{z_1} J^{w_1}$, whereas server can compute X_1 as K^{r_2} and X_2 as $F^{x_1} G^{y_1} (I/\mathsf{pw})^{z_1} J^{w_1}$.

Security. To perform authentication, client and server need to (implicitly) prove to each other that they know pw. This is achieved as follows. Note that $X_1 = A^{x_2} B^{y_2} (C/\mathsf{pw})^{z_2} D^{w_2}$ has the following property: given $E = g_1^{x_2} g_2^{y_2} h^{z_2} (cd^{\alpha})^{w_2}$ (but not x_2, y_2, z_2, w_2), if (A, B, C, D) is a valid encryption of pw, then knowing the randomness r_1 is sufficient for computing X_1 (as E^{r_1}); otherwise X_1 is a uniformly random group element.[8] Therefore, an adversarial client that does not know pw, is not able to come up with a valid (A, B, C, D), so X_1 is uniformly random in the adversary's view (and so is $\mathsf{sk} = X_1 X_2$); a symmetric argument can be made for an adversarial server. In the man-in-the-middle setting, an adversary can attempt to generate a valid ciphertext after seeing another valid ciphertext from the honest client/server, so we need the encryption scheme to be non-malleable (hence ElGamal does not suffice).

The one-time signature scheme effectively forces the adversary to forward msg_2 and msg_3 without modification as long as it forwards msg_1. This is to prevent a man-in-the-middle adversary from gaining information by forwarding all ciphertexts but modifying the rest of the messages. Specifically, (if the signature is removed) consider an adversary that forwards $\mathsf{msg}_1 = (A, B, C, D)$, changes $\mathsf{msg}_2 = (E, F, G, I, J)$ to $(E^{\frac{1}{2}}, F, G, I, J)$, and changes $\mathsf{msg}_3 = K$ to K^2; this would cause $\mathsf{sk}_{\mathsf{server}} = \mathsf{sk}_{\mathsf{client}}^2$. In other words, the adversary (that does not know the password) causes the two parties' session keys to be unequal but correlated, which is not allowed by the security of PAKE. Furthermore, to prevent the adversary from changing VK to its own verification key (and thus knowing the corresponding signing key), VK is included in the hash that produces α. In this way if the adversary changes VK while keeping the ciphertext (A, B, C, D), the ciphertext would become invalid.

1.2 Technical Overview

In this section, we provide a high-level explanation of why the KOY protocol is insecure in the UC framework, and how the AGM circumvents the difficulty for the UC simulator. We assume the reader is familiar with the UC PAKE functionality (Fig. 2).

UC-insecurity of KOY. Our attack relies on an adversary $\mathcal{A}$ that completely disregards the presence of server and instead interacts with client while executing server's algorithm on its own. In particular, once the protocol is initiated by client, $\mathcal{A}$ assumes the role of the server (discarding the actual server in the process, which plays no part in the protocol) and receives $\mathsf{msg}_1 =$

[8] Using the terminology of SPHF: r_1 is the witness for the language, (x_2, y_2, z_2, w_2) is the hash key, and E is the corresponding projection key; the hash function is defined as $\mathsf{Hash}_{(x_2,y_2,z_2,w_2)}(\mathsf{pw}) = A^{x_2} B^{y_2} (C/\mathsf{pw})^{z_2} D^{w_2}$ and the projection function is defined as $\mathsf{HashProj}_E(r_1) = E^{r_1}$.

$(\mathsf{VK}, A, B, C, D)$. After this, $\mathcal{A}$ runs the server's algorithm on client's password pw (i.e., we assume that $\mathcal{A}$ makes a correct password guess) and computes $\mathsf{msg}_2 = (E, F, G, I, J)$. client then runs its session-key generating algorithm and outputs its session key $\mathsf{sk} = E^{r_1} F^{x_1} G^{y_1} (I/\mathsf{pw})^{z_1} J^{w_1}$ to the environment $\mathcal{Z}$. At this point, $\mathcal{A}$ (and $\mathcal{Z}$) have all the information they need to run the server's session-key generating algorithm locally, which computes a session key equal to sk output by client.

To see why a simulator $\mathcal{S}$ cannot simulate this adversary, we attempt an ideal-world execution and pinpoint where it fails. Since $\mathcal{S}$ is allowed to choose $\mathsf{crs} = (g_1, g_2, h, c, d)$, it can sample g_1 at random and set h such that $h = g_1^\kappa$ — in other words, $\mathcal{S}$ chooses the "CRS trapdoor" $\kappa = \log_{g_1} h$. After receiving the NewSession command from the UC PAKE functionality $\mathcal{F}_{\mathsf{pake}}$ (indicating the start of the session), $\mathcal{S}$ must simulate client's first message msg_1. Since $\mathcal{S}$ does not know the password pw at this point, it must (effectively) guess some $\widetilde{\mathsf{pw}}$ at random; that is, in msg_1, $C = h^{r_1} \cdot \widetilde{\mathsf{pw}}$ where $\widetilde{\mathsf{pw}}$ can be no better than a random password sampled from the dictionary. (C is indistinguishable from the correct value due to the security of Cramer–Shoup encryption.) After $\mathcal{Z}$ responds with $\mathsf{msg}_2 = (E, F, G, I, J)$, since $F = g_1^{r_2}$ and $I = h^{r_2} \cdot \mathsf{pw}$, $\mathcal{S}$ can extract pw as I/F^κ. Once this has been done, $\mathcal{S}$ can send a TestPwd command to $\mathcal{F}_{\mathsf{pake}}$ on the correct pw; $\mathcal{F}_{\mathsf{pake}}$ would mark the client session compromised and thus allow $\mathcal{S}$ to choose client's session key sk (which has to be consistent with the session key client computes in the real world).[9] This is where the game-based security and UC-security of PAKE diverge: in game-based security, all security guarantees are considered lost (and the simulation of the game can stop) once the adversary guesses the correct password; whereas in UC-security the simulation has to continue. The problem here is that *even knowing the correct password* pw*,* $\mathcal{S}$ *still cannot determine what* sk *should be.*

Recall that sk is the product of

$$X_1 = E^{r_1} = A^{x_2} B^{y_2} \left(\frac{C}{\mathsf{pw}}\right)^{z_2} D^{w_2}$$

and

$$X_2 = K^{r_2} = F^{x_1} G^{y_1} \left(\frac{I}{\mathsf{pw}}\right)^{z_1} J^{w_1}.$$

Computing X_2 is not a problem for $\mathcal{S}$, since $\mathsf{msg}_2 = (E, F, G, I, J)$ is provided to $\mathcal{S}$ directly from the environment; $\mathcal{S}$ chose x_1, y_1, z_1, w_1 on its own; and $\mathcal{S}$ has extracted pw. However, $\mathcal{S}$ is not able to compute X_1. At first glance, computing $X_1 = E^{r_1}$ might appear feasible as $\mathcal{S}$ received E as part of msg_2 and sampled r_1 before sending msg_1. However, the problem is that *the password guess* $\widetilde{\mathsf{pw}}$ *that* $\mathcal{S}$ *uses while generating* msg_1 *is likely incorrect*; as a result, E^{r_1} that $\mathcal{S}$ computes is actually equal to $A^{x_2} B^{y_2} (C/\widetilde{\mathsf{pw}})^{z_2} D^{w_2}$, whereas the correct value should be $A^{x_2} B^{y_2} (C/\mathsf{pw})^{z_2} D^{w_2}$. This means that $\mathcal{S}$ must know $(\mathsf{pw}/\widetilde{\mathsf{pw}})^{z_2}$ in order to compute the correct sk, which is infeasible since z_2 was chosen by $\mathcal{Z}$ and unknown to $\mathcal{S}$ (unless $\widetilde{\mathsf{pw}} = \mathsf{pw}$, which happens with probability at most $1/|\mathcal{PW}|$ where $\mathcal{PW}$ is the password dictionary).[10]

Simulating the session key in AGM. Our next critical observation is that the session key in the above attack can be simulated if we resort to the AGM, as the simulator $\mathcal{S}$ can extract z_2

[9] A TestPwd *must* be run, since otherwise $\mathcal{F}_{\mathsf{pake}}$ would mark client's session fresh, resulting in $\mathcal{F}_{\mathsf{pake}}$ outputting a uniformly random key to client.

[10] Using the terminology of SPHF: Since (A, B, C, D) is likely not an encryption of pw, the corresponding SPHF value $X_1 = \mathsf{Hash}_{(x_2, y_2, z_2, w_2)}(\mathsf{pw})$ is independent of $\mathcal{S}$'s view due to the (perfect) smoothness property of the SPHF.

from an algebraic environment. In more detail, suppose $\mathcal{Z}$ runs the above attack and sends E as part of msg_2. At this point all group elements that $\mathcal{Z}$ has seen are g_1, g_2, h, c, d from crs and A, B, C, D from msg_1. An algebraic $\mathcal{Z}$ must "explain" how E is computed; for now let's ignore A, B, C, D and assume $\mathcal{Z}$ computes

$$E = g_1^{x_2} g_2^{y_2} h^{z_2} (cd^{\alpha})^{w_2}.$$

In the real world client would compute $X_1 = E^{r_1}$, which is equal to $A^{x_2} B^{y_2} (C/\mathsf{pw})^{z_2} D^{w_2}$. In the ideal world, as explained above, $\mathcal{S}$ cannot compute X_1 as E^{r_1} since it likely chose a wrong $\widetilde{\mathsf{pw}}$ while generating A, B, C, D, causing E^{r_1} that $\mathcal{S}$ computes to be equal to $A^{x_2} B^{y_2} (C/\widetilde{\mathsf{pw}})^{z_2} D^{w_2}$. However, after extracting the correct pw from msg_2 (as I/F^{κ}), $\mathcal{S}$ can "correct its previous mistake" by computing X_1 as $E^{r_1}(\widetilde{\mathsf{pw}}/\mathsf{pw})^{z_2}$, since now it sees the algebraic coefficient z_2 from $\mathcal{Z}$. In this way $\mathcal{S}$ generates X_1 that is indistinguishable from the real world.

In the general case, suppose $\mathcal{Z}$ computes

$$E = g_1^{x_2} g_2^{y_2} h^{z_2} c^{w_2} d^{v_2} A^{x'_2} B^{y'_2} C^{z'_2} D^{w'_2}.$$

A very similar analysis to above shows that $\mathcal{S}$ can still compute X_1 as $E^{r_1}(\widetilde{\mathsf{pw}}/\mathsf{pw})^{z_2}$; the key point is that given (A, B, C, D), the only place where the "wrong" password $\widetilde{\mathsf{pw}}$ is used lies in h, so we only need to "correct" h^{r_1} from $C/\widetilde{\mathsf{pw}}$ to C/pw (and the additional terms involving A, B, C, D, pw are irrelevant).

Further subtleties in UC. As we have just seen, one critical difference between the game-based and UC security notions for PAKE is that in UC the real view and the simulated view must remain indistinguishable *even when the environment sees the key of a successfully attacked session*, whereas in the game-based setting the adversary simply wins (and the simulator can "give up" on simulating the session key) once the attack on a session is successful. While the session key itself can be simulated in the AGM, this poses another potential issue that is subtler: after seeing the session key, the environment might go back and check the validity of *previous protocol messages* using this information.

In more detail, assume again that the adversary interacts with client by running server's algorithm on client's password pw. When client's session completes, the environment $\mathcal{Z}$ sees client's session key $\mathsf{sk}_{\mathsf{client}} = X_1 X_2$ and the last message $\mathsf{msg}_3 = (K, \sigma)$. Since $\mathcal{Z}$ can compute X_2 as K^{r_2} (since r_2 is chosen by $\mathcal{Z}$ itself), it can recover $X_1 = E^{r_1}$ (where E is part of msg_2 chosen by $\mathcal{Z}$ and r_1 is the randomness used in msg_1) as $\mathsf{sk}_{\mathsf{client}}/X_2$. Recall that in the ideal world the Cramer–Shoup ciphertext (A, B, C, D) generated by the simulator $\mathcal{S}$ is an encryption of some pw' that is unlikely to be the "correct" pw, so $\mathcal{Z}$ must not be able to detect this fact even after seeing E^{r_1}. In other words, (very roughly) *Cramer–Shoup must be secure even if the adversary additionally sees E^{r_1} for some E of its choice.*

Below we analyze three simple attacks using this strategy:

1. Say $\mathcal{Z}$ chooses $E = A = g_1^{r_1}$; then $X_1 = E^{r_1} = g_1^{r_1^2}$. If 2-DL is easy in the group[11], then $\mathcal{Z}$ can recover r_1 after client's session completes, and check if $C = h^{r_1} \cdot \mathsf{pw}$. In the real world this equation holds, whereas in the ideal world it does not if $\mathcal{S}$ chose the "wrong" $\widetilde{\mathsf{pw}} \neq \mathsf{pw}$ to encrypt while simulating msg_1. In fact, it seems that for the KOY protocol

[11]The 2-DL problem is: given (g^x, g^{x^2}) for $x \xleftarrow{\$} \mathbb{Z}_q$, compute x. We do not know the relative hardness between 2-DL and DDH, and it has been shown that 2-DL and CDH are separate in the AGM [BFL20].

to be UC-AGM-secure, $X_1 = g_1^{r_1^2}$ needs to leak no information about r_1. This is the stronger *Decisional Square Diffie–Hellman (DSDH)* assumption, which says that given g^x for $x \leftarrow \mathbb{Z}_q$, it is hard to distinguish g^{x^2} from a random group element.[12]

2. Say $\mathcal{Z}$ chooses $E = c$; then $X_1 = E^{r_1} = c^{r_1}$. But Cramer–Shoup is obviously *not* CCA-secure if the adversary additionally sees c^{r_1}. What saves us here is that $\mathcal{Z}$ sees E^{r_1} *only at the end of* client*'s session.* Indeed, the reduction $\mathcal{R}$ to the security of Cramer–Shoup roughly works as follows:

   (a) $\mathcal{R}$ embeds the target ciphertext as msg_1, client's first message intercepted by $\mathcal{A}$;

   (b) When $\mathcal{A}$ sends msg_1^* to server and msg_2^* to client, $\mathcal{R}$ needs to query the decryption oracle to extract the password guesses contained in these two messages;

   (c) Finally, if the password guess in msg_2^* is correct, $\mathcal{R}$ needs to simulate $\mathsf{sk}_{\mathsf{client}}$. (This step is not needed in the game-based proof).[13]

   c^{r_1} is needed only in step (c), which happens after step (b) when all decryption oracle queries are made. Therefore, we only need Cramer–Shoup to remain CCA-secure if the adversary *is restricted to making all decryption oracle queries before learning* c^{r_1}. We will show that Cramer–Shoup indeed satisfies this property — which we call *delayed-reveal-CCA security* — in Section A.

3. A slightly confusing case is that $\mathcal{Z}$ chooses $E = h$. It appears that upon seeing $X_1 = E^{r_1} = h^{r_1}$, $\mathcal{Z}$ can compute $\widetilde{\mathsf{pw}}$ as C/X_1 and check if $\widetilde{\mathsf{pw}} = \mathsf{pw}$; in the real world this equation holds, whereas in the ideal world the simulator $\mathcal{S}$ would be "caught" using a wrong $\widetilde{\mathsf{pw}}$. However, this case can in fact be simulated perfectly. As mentioned above, $\mathcal{S}$ is able to "correct its previous mistake" by simulating X_1 as $C/\mathsf{pw} = h^{r_1} \cdot (\widetilde{\mathsf{pw}}/\mathsf{pw})$ instead of h^{r_1}, and if $\mathcal{Z}$ computes C/X_1, the result would be exactly pw (rather than $\widetilde{\mathsf{pw}}$). In general, the h term in the algebraic expression of E is a special case where $\mathcal{S}$ needs to use $h^{r_1} \cdot (\widetilde{\mathsf{pw}}/\mathsf{pw})$ in place of h^{r_1}, i.e., "pretend" that h^{r_1} is C/pw rather than $C/\widetilde{\mathsf{pw}}$, in order to match the real-world view.

By inspecting the game-based security proof of the KOY protocol, one can see that the above cases (and their combination) essentially cover all additional attacks the environment can make in the UC setting, and the security argument for all other cases (including the adversary forwarding a message, sending a Cramer–Shoup ciphertext that contains an incorrect password guess, or modifying the signature) is essentially identical to the game-based proof.

2 Preliminaries

2.1 Cryptographic Assumptions

Let λ be the security parameter. Let $\mathcal{G}$ be a group generation algorithm that on input 1^λ outputs $\Gamma[\mathbb{G}, g, q]$, where $\mathbb{G}$ is a group of prime order q with $2^\lambda < q < 2^{\lambda+1}$, and g is a generator of $\mathbb{G}$. The

[12] DSDH implies 2-DL as given (g^x, Y) where Y is either g^{x^2} or a uniformly random group element, a DSDH distinguisher can feed (g^x, Y) to the 2-DL solver and distinguish by observing whether the 2-DL solver succeeds or not.

[13] We omit the remaining steps which are to simulate msg_3, and on $\mathcal{A}$'s message msg_3^* simulate server's session key $\mathsf{sk}_{\mathsf{server}}$, as they are inconsequential to our main point.

set of all possible outputs of $\mathcal{G}(1^\lambda)$ is denoted $[\mathcal{G}_\lambda]$. We assume that group operations in $\mathbb{G}$ can be done in polynomial time in λ, including exponentiation, sampling a random group element, and membership check. For simplicity, we may drop 1^λ and Γ from the inputs of algorithms and do not explicitly write them.

We define $\bar{\mathbb{G}}$ to be $\mathbb{G} \setminus \{1\}$; since q is prime, $\bar{\mathbb{G}}$ is the set of generators of $\mathbb{G}$. If we sample $g \xleftarrow{\$} \bar{\mathbb{G}}$, then we obtain a random generator.

Below we define two assumptions we use in this work, the *Decisional Diffie–Hellman (DDH) assumption* and the *Decisional Square Diffie–Hellman (DSDH) assumption.*

The DDH and DSDH assumption. Let $\mathcal{G}$ be as above. For any Probabilistic Polynomial-Time (PPT) distinguisher $\mathcal{A}$, we define the DDH advantage of $\mathcal{A}$ as

$$\mathsf{AdvDDH}_{\mathcal{G},\mathcal{A}}(\lambda|\Gamma) = \left|\Pr[\mathcal{A}(g^x, g^y, g^{xy}) = 1 : x, y \xleftarrow{\$} \mathbb{Z}_q] - \Pr[\mathcal{A}(g^x, g^y, g^z) = 1 : x, y \xleftarrow{\$} \mathbb{Z}_q, z \xleftarrow{\$} \mathbb{Z}_q \setminus \{xy\}]\right|$$

and define the DSDH advantage of $\mathcal{A}$ as

$$\mathsf{AdvDSDH}_{\mathcal{G},\mathcal{A}}(\lambda|\Gamma) = \left|\Pr[\mathcal{A}(g^x, g^{x^2}) = 1 : x \xleftarrow{\$} \mathbb{Z}_q] - \Pr[\mathcal{A}(g^x, g^y) = 1 : x, y \xleftarrow{\$} \mathbb{Z}_q]\right|.$$

The DDH (resp. DSDH) assumption for $\mathcal{G}$ states that for every PPT distinguisher $\mathcal{A}$, there exists some negligible function negl such that for all $\lambda \in \mathbb{Z}_{>0}$, $\mathsf{AdvDDH}_{\mathcal{G},\mathcal{A}}(\lambda|\Gamma) \leq \mathsf{negl}(\lambda)$ (resp. $\mathsf{AdvDSDH}_{\mathcal{G},\mathcal{A}}(\lambda|\Gamma) \leq \mathsf{negl}(\lambda)$).

Note that in our definition of DDH, the random g^z must not equal to g^{xy}. This matches the use of DDH in our proof. Obviously it is equivalent to the standard version where g^z is sampled uniformly at random from $\mathbb{G}$.

Relation between DDH and DSDH. It is known that the DSDH assumption implies the DDH assumption; we give a self-contained proof here.

Lemma 2.1. *If the DSDH assumption holds for a group generation algorithm $\mathcal{G}$, then the DDH assumption holds for $\mathcal{G}$.*

Proof. Given a DDH distinguisher $\mathcal{A}_{\mathsf{DDH}}$, we construct a DSDH distinguisher $\mathcal{A}_{\mathsf{DSDH}}$ as follows. Given input (X, Y), $\mathcal{A}_{\mathsf{DSDH}}$ samples a random $r \xleftarrow{\$} \mathbb{Z}_q$ and runs $\mathcal{A}_{\mathsf{DDH}}$ on $(X, X \cdot g^r, Y \cdot X^r)$. $\mathcal{A}_{\mathsf{DSDH}}$ then outputs what $\mathcal{A}_{\mathsf{DDH}}$ outputs.

Suppose $X = g^x$. If $Y = g^{x^2}$, then $\mathcal{A}_{\mathsf{DDH}}$'s input is $(g^x, g^{x+r}, g^{x^2+rx})$ which is a DH tuple, with g^x and g^{x+r} being two independent uniformly random elements in $\mathbb{G}$. If Y is a random group element independent of X, then $\mathcal{A}_{\mathsf{DDH}}$'s input is a uniformly random 3-tuple in $\mathbb{G}^3$. (Here we use the "standard" version of DDH where g^z is uniformly random in $\mathbb{G}$, rather than $\mathbb{G} \setminus \{g^{xy}\}$.) It follows that the distinguishing advantage of $\mathcal{A}_{\mathsf{DSDH}}$ is equal to that of $\mathcal{A}_{\mathsf{DDH}}$, completing the proof. □

Remark 2.1. *Bao, Deng and Zhu [BDZ03, Section 3.2] give an alternative proof of Lemma 2.1, where $\mathcal{A}_{\mathsf{DSDH}}$ samples random $r, s \xleftarrow{\$} \mathbb{Z}_q$ and runs $\mathcal{A}_{\mathsf{DDH}}$ on (X^r, X^s, Y^{rs}) — that is, the first two group elements are both randomized. In fact it is sufficient to randomize only one of them, making the reduction slightly more efficient.*

2.2 Cryptographic Primitives

2.2.1 One-Time Signature Schemes

A *one-time signature scheme* OTS is a triple of PPT algorithms $(\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Vrfy})$ such that:

- The key generation algorithm KeyGen takes as input the security parameter 1^λ and outputs a key pair $(\mathsf{VK}, \mathsf{SK})$.
- The signing algorithm Sign takes as input SK and a message m and returns a signature σ, denoted as $\sigma \leftarrow \mathsf{Sign}_{\mathsf{SK}}(m)$.
- The (deterministic) verification algorithm Vrfy takes as input a verification key VK, a message m, and a signature σ and returns a bit b, denoted as $b := \mathsf{Vrfy}_{\mathsf{VK}}(m; \sigma)$.

We require the scheme to satisfy the following properties:

- **Correctness**: For all $\lambda \in \mathbb{Z}_{\geq 0}$, if $(\mathsf{VK}, \mathsf{SK}) \leftarrow \mathsf{KeyGen}(1^\lambda)$, then for all m and all $\sigma \leftarrow \mathsf{Sign}_{\mathsf{SK}}(m)$, we have $\mathsf{Vrfy}_{\mathsf{VK}}(m; \sigma) = 1$ with probability 1.
- **Strong one-time security:** Let $\mathcal{A}$ be any PPT algorithm. For $\lambda \in \mathbb{Z}_{\geq 0}$, we define the OTS advantage of $\mathcal{A}$ as

  $$\mathsf{AdvOTS}_{\mathsf{OTS},\mathcal{A}}(\lambda) := \Pr[\mathsf{Vrfy}_{\mathsf{VK}}(m; \sigma) = 1 : (\mathsf{VK}, \mathsf{SK}) \leftarrow \mathsf{KeyGen}(1^\lambda), (m, \sigma) \leftarrow \mathcal{A}^{\mathsf{Sign}_{\mathsf{SK}}(\cdot)}(\mathsf{VK})]$$

  where $\mathcal{A}$ makes only a single query to $\mathsf{Sign}_{\mathsf{SK}}(\cdot)$ and σ was not an output of $\mathsf{Sign}_{\mathsf{SK}}(m)$. We say that OTS is strongly one-time secure if there exists some negligible function negl such that for all PPT $\mathcal{A}$ and $\lambda \in \mathbb{Z}_{\geq 0}$,

  $$\mathsf{AdvOTS}_{\mathsf{OTS},\mathcal{A}}(\lambda) \leq \mathsf{negl}(\lambda).$$

2.2.2 Collision-Resistant Hash Functions

Informally, a function H is collision-resistant if no PPT algorithm can find $x \neq x'$ such $H(x) = H(x')$ with higher than negligible probability. Formally, we define a hashing scheme HF associated with $\mathcal{G}$ to specify (a) a family of key spaces $\mathsf{HF.Keyspace}_{\lambda,\Gamma}$ indexed by $\lambda \in \mathbb{Z}_{\geq 0}$ and $\Gamma \in [S_\lambda]$, and (b) a family of hash functions indexed by $\lambda \in \mathbb{Z}_{\geq 0}$, $\Gamma \in [\mathcal{G}_\lambda]$ and $\mathsf{hk} \in [\mathsf{HF.Keyspace}_{\lambda,\Gamma}]$ where each $\mathsf{HF}_{\mathsf{hk}}^{\lambda,\Gamma}$ maps a tuple ρ of group elements to an element of $\mathbb{Z}_q$. Furthermore, the algorithm that outputs $\mathsf{HF}_{\mathsf{hk}}^{\lambda,\Gamma}(\rho)$ must be deterministic and polynomial-time.

Let $\mathcal{A}$ be any PPT algorithm. For $\lambda \in \mathbb{Z}_{\geq 0}$, we define the CRH advantage of $\mathcal{A}$ given Γ as

$$\mathsf{AdvCRH}_{\mathsf{HF},\mathcal{A}}(\lambda|\Gamma) := \Pr[x \neq x' \wedge \mathsf{HF}_{\mathsf{hk}}^{\lambda,\Gamma}(x) = \mathsf{HF}_{\mathsf{hk}}^{\lambda,\Gamma}(x') : \mathsf{hk} \xleftarrow{\$} \mathsf{HF.Keyspace}_{\lambda,\Gamma}, (x, x') \leftarrow \mathcal{A}(\mathsf{hk})]$$

where $\Gamma \xleftarrow{\$} [\mathcal{G}_\lambda]$. We say that HF is collision-resistant if there exists some negligible function negl such that for all PPT $\mathcal{A}$ and $\lambda \in \mathbb{Z}_{\geq 0}$,

$$\mathsf{AdvCRH}_{\mathsf{HF},\mathcal{A}}(\lambda|\Gamma) \leq \mathsf{negl}(\lambda).$$

Key Generation: On input 1^λ for $\lambda \in \mathbb{Z}_{\geq 0}$, compute

$$\Gamma[\mathbb{G}, g, q] \leftarrow \mathcal{G}(1^\lambda); \mathsf{hk} \xleftarrow{\$} \mathsf{HF.Keyspace}_{\lambda,\Gamma}$$
$$w \xleftarrow{\$} \mathbb{Z}_q^*; x_1, x_2, y_1, y_2, z_1, z_2 \xleftarrow{\$} \mathbb{Z}_q$$
$$g_1 := g; g_2 := g^w; c := g_1^{x_1} g_2^{x_2}; d := g_1^{y_1} g_2^{y_2}; h := g_1^{z_1} g_2^{z_2}$$

and output the public key $\mathsf{PK} := (\Gamma, \mathsf{hk}, g_1, g_2, c, d, h)$ and the secret key $\mathsf{SK} := (\Gamma, \mathsf{hk}, x_1, x_2, y_1, y_2, z_1, z_2)$.

Encryption: Given a public key

$$\mathsf{PK} = (\Gamma, \mathsf{hk}, g_1, g_2, c, d, h) \in [\mathcal{G}_\lambda] \times [\mathsf{HF.Keyspace}_{\lambda,\Gamma}] \times \mathbb{G}^5,$$

a message $m \in \mathbb{G}$ and a label label, compute

E1: $r \xleftarrow{\$} \mathbb{Z}_q$;
E2: $u_1 := g_1^r$;
E3: $u_2 := g_2^r$;
E4: $e' := h^r$;
E5: $e := e' \cdot m$;
E6: $\alpha := \mathsf{HF}_{\mathsf{hk}}^{\lambda,\Gamma}(\mathsf{label}, u_1, u_2, e)$;
E7: $v := (cd^\alpha)^r$;

and output the ciphertext $\psi := (u_1, u_2, e, v)$.

Decryption: Given a secret key

$$\mathsf{SK} = (\Gamma, \mathsf{hk}, x_1, x_2, y_1, y_2, z_1, z_2) \in [\mathcal{G}_\lambda] \times [\mathsf{HF.Keyspace}_{\lambda,\Gamma}] \times \mathbb{Z}_q^6,$$

along with a ciphertext ψ and a label label, do the following.

D1: Parse ψ as a 4-tuple $(u_1, u_2, e, v) \in \mathbb{G}^4$; output $\perp$ and halt if ψ is not of this form.
D2: Test if $u_1, u_2, e \in \mathbb{G}$; output $\perp$ and halt if this is not the case.
D3: Compute $v' := \mathsf{HF}_{\mathsf{hk}}^{\lambda,\Gamma}(\mathsf{label}, u_1, u_2, e)$.
D4: Test if $v = u_1^{x_1+y_1\alpha} u_2^{x_2+y_2\alpha}$; output $\perp$ and halt if this is not the case.
D5: Compute $e' := u_1^{z_1} u_2^{z_2}$.
D6: Output $m := e/e'$.

Figure 1: The Cramer–Shoup Encryption Scheme with Labels.

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

  Finally, mark the record completed.
- On $(\mathsf{Abort}, \mathsf{sid}, \mathsf{P})$ from $\mathcal{S}$, if there is a record $\langle \mathsf{P}, \mathsf{P}', \mathsf{pw} \rangle$ not marked completed, output $(\mathsf{sid}, \perp)$ to P and mark the record completed.

Figure 2: UC PAKE Functionality $\mathcal{F}_{\mathsf{pake}}$.

2.3 Cramer–Shoup Encryption Scheme

The KOY protocol uses the Cramer–Shoup public-key encryption scheme, which has been proven CCA-secure under the DDH assumption. In fact, we will use a modification of the Cramer–Shoup scheme that supports *labels* [KOY01, Sho01].

We provide a formal description of the scheme in Fig. 1. Apart from the (standard) CCA-security of the scheme, we rely on a stronger variant of security we term *delayed-reveal-CCA-security (DR-CCA-security)*, in which the adversary is revealed certain (potentially sensitive) information *after* all decryption queries have been made. This modified security property is crucial to our security proof (see Section 1.2 for an overview). We postpone discussion of DR-CCA security to Section A.

2.4 UC PAKE

We recall the standard UC PAKE functionality [CHK+05] in Fig. 2. A more detailed explanation can be found in [RX23, Section 2.2].

We describe below how adversarial attacks are modelled within the scope of the functionality, which is crucial to our security analysis.

- In the case of an *eavesdropping attack*, both instances P and P' are marked as fresh. If $\mathsf{pw} = \mathsf{pw}'$, i.e., the two parties' passwords match, then the protocol execution in the real world is correct, and no TestPwd command is sent to $\mathcal{F}_{\mathsf{pake}}$. In this case, a session key sk is sampled first *uniformly*, and the other session key sk' is set to be equal to sk. This corresponds to a correct protocol execution without adversary interference.

On the other hand, if $\mathsf{pw} \neq \mathsf{pw}'$, then again no TestPwd is sent. In this case, the two parties' keys sk and sk' are independently random.

- In the case of a successful online guessing attack, the simulator sends a TestPwd command and receives "correct guess" while the session is marked as compromised. The simulator then submits a key sk^* (which is output in the real protocol and can be determined from the actions of the adversary) to $\mathcal{F}_{\mathsf{pake}}$ which sets the key of the party to sk^*. *Note that this is a significant departure from the game-based security definition of PAKE.* In the game-based security definition, all security guarantees are lost in the case of a correct password guess, and the simulation can be terminated (by outputting a "dummy" symbol ∇, as in [KOY01]). In the UC-security definition, simulation must continue, and the simulator must be able to determine the attacked party's output key sk^* from the adversary's messages.

- In the case of a failed online guessing attack by the adversary, the session is marked as interrupted. In this case, the session keys of P and P' are independently random, and the simulator gains no knowledge of either.

- If a session is unattacked (i.e., fresh) but its counterparty's session is attacked (no matter whether successful or not), then the simulator should gain no information about the session key of P, and it must be output uniformly at random.

Our proof of security relies heavily on simulating each of the above cases.

We also add an Abort command to the functionality, which will cause a party to output $\perp$ and immediately terminate its session. This is necessary since the server in the KOY protocol outputs $\perp$ if the one-time signature is invalid. Finally, we require the session key sk to be a group element rather than a λ-bit string, again in order to be consistent with the KOY protocol.

3 Proof of UC-Insecurity

3.1 Formal Description of the Protocol

We provide a formal description of the PAKE protocol of KOY in Fig. 3, adapted to UC formalism. (For a high-level description, see Section 1.1.) We assume w.l.o.g. that the session ID sid contains the names of the protocol parties.

It is easy to see that the protocol is perfectly correct, i.e., if there is no adversary then client and server output the same session key $\mathsf{sk}_{\mathsf{client}} = \mathsf{sk}_{\mathsf{server}}$.

3.2 UC-Insecurity of KOY

Theorem 3.1. *The protocol in Fig. 3 does not UC-realize $\mathcal{F}_{\mathsf{pake}}$.*

Proof. Consider the environment $\mathcal{Z}$ in Figure 4 and the dummy adversary. It follows from the correctness of the protocol that in the real-world protocol execution $\mathcal{Z}$ always outputs 1, since the algorithm of $\mathcal{Z}$ and $\mathcal{A}$ is the same as that of an honest server. We will show that for any simulator $\mathcal{S}$ in the ideal world, the probability that $\mathcal{Z}$ outputs 1 is roughly $1/|\mathcal{PW}|$ (even if $\mathcal{S}$ is computationally unbounded).

First consider the moment when $\mathcal{S}$ sends $\mathsf{msg}_1 = (\mathsf{sid}, \mathsf{VK}, A, B, C, D)$ to $\mathcal{Z}$ (in step 2). At this point the only message $\mathcal{S}$ has seen is $(\mathsf{NewSession}, \mathsf{sid}, \mathsf{client}, \mathsf{server})$ from $\mathcal{F}_{\mathsf{pake}}$ (in step 1), so pw is independent of $\mathcal{S}$'s view and the probability that $\mathsf{pw} = C/A^{\log h}$ is at most $1/|\mathcal{PW}|$.

Initialization: Let $\Gamma[\mathbb{G}, g, q] \leftarrow \mathcal{G}(1^\lambda)$. Set $g_1 := g$ and sample $g_2, c, d, h \xleftarrow{\$} \bar{\mathbb{G}}$. Furthermore, sample $\mathsf{hk} \xleftarrow{\$} \mathsf{HF.Keyspace}_{\lambda,\Gamma}$. Output $\mathsf{crs} := (\Gamma, g_1, g_2, h, c, d, \mathsf{hk})$ as the public parameter.

Protocol Execution: We describe the execution of the protocol below.

1. **Round $\mathbf{R}_1$.** On input $(\mathsf{NewSession}, \mathsf{sid}, \mathsf{client}, \mathsf{server}, \mathsf{pw}_{\mathsf{client}}, \text{“client”})$, client runs $(\mathsf{VK}, \mathsf{SK}) \leftarrow \mathsf{OTS.Gen}(1^\lambda)$ and samples $r_1 \xleftarrow{\$} \mathbb{Z}_q$. It then sets

$$\begin{aligned} A &:= g_1^{r_1} \\ B &:= g_2^{r_1} \\ C &:= h^{r_1} \cdot \mathsf{pw}_{\mathsf{client}} \\ \alpha &:= \mathsf{HF}_{\mathsf{hk}}^{\lambda,\Gamma}(\mathsf{sid}, \mathsf{VK}, A, B, C) \\ D &:= (cd^\alpha)^{r_1} \end{aligned}$$

   and sends $\mathsf{msg}_1 := (\mathsf{sid}, \mathsf{VK}, A, B, C, D)$.

2. **Round $\mathbf{R}_2$.** On input $(\mathsf{NewSession}, \mathsf{sid}, \mathsf{client}, \mathsf{server}, \mathsf{pw}_{\mathsf{server}}, \text{“server”})$ and $\mathsf{msg}_1 = (\mathsf{sid}, \mathsf{VK}, A, B, C, D)$ from client, server samples $x_2, y_2, z_2, w_2, r_2 \xleftarrow{\$} \mathbb{Z}_q$ and sets

$$\begin{aligned} \alpha' &:= \mathsf{HF}_{\mathsf{hk}}^{\lambda,\Gamma}(\mathsf{sid}, \mathsf{VK}, A, B, C) \\ E &:= g_1^{x_2} g_2^{y_2} h^{z_2} (cd^{\alpha'})^{w_2} \\ F &:= g_1^{r_2} \\ G &:= g_2^{r_2} \\ I &:= h^{r_2} \cdot \mathsf{pw}_{\mathsf{server}} \\ \beta &:= \mathsf{HF}_{\mathsf{hk}}^{\lambda,\Gamma}(\mathsf{msg}_1, \mathsf{sid}, E, F, G, I) \\ J &:= (cd^\beta)^{r_2} \end{aligned}$$

   and sends $\mathsf{msg}_2 := (\mathsf{sid}, E, F, G, I, J)$.

3. **Round $\mathbf{R}_3$ (Client).** On $\mathsf{msg}_2 = (\mathsf{sid}, E, F, G, I, J)$ from server, client samples $x_1, y_1, z_1, w_1 \xleftarrow{\$} \mathbb{Z}_q$ and sets

$$\begin{aligned} \beta' &:= \mathsf{HF}_{\mathsf{hk}}^{\lambda,\Gamma}(\mathsf{sid}, \mathsf{msg}_1, E, F, G, I) \\ K &:= g_1^{x_1} g_2^{x_2} h^{z_1} (cd^{\beta'})^{w_1} \\ \sigma &\leftarrow \mathsf{Sign}_{\mathsf{SK}}(\mathsf{msg}_1, \mathsf{msg}_2, K) \end{aligned}$$

   and sends $\mathsf{msg}_3 := (\mathsf{sid}, K, \sigma)$ to server. It then computes $I' := I/\mathsf{pw}_{\mathsf{client}}$ and outputs

$$(\mathsf{sid}, \mathsf{sk}_{\mathsf{client}} := E^{r_1} F^{x_1} G^{y_1} (I')^{z_1} J^{w_1}).$$

4. **Round $\mathbf{R}_3$ (Server).** On $\mathsf{msg}_3 = (\mathsf{sid}, K, \sigma)$ from client, server checks if $\mathsf{Vrfy}_{\mathsf{VK}}(\mathsf{msg}_1, \mathsf{msg}_2, K; \sigma) = 1$. If not, it outputs $(\mathsf{sid}, \perp)$. Else, it computes $C' := C/\mathsf{pw}_{\mathsf{server}}$ and outputs

$$(\mathsf{sid}, \mathsf{sk}_{\mathsf{server}} := A^{x_2} B^{y_2} (C')^{z_2} D^{w_2} K^{r_2}).$$

Figure 3: PAKE Protocol of KOY.

0. $\mathcal{Z}$ receives the public parameter $\mathsf{crs} = (\Gamma, g_1, g_2, h, c, d, \mathsf{hk})$.

1. $\mathcal{Z}$ samples $\mathsf{pw} \xleftarrow{\$} \mathcal{PW}$, where $\mathcal{PW} \subseteq \mathbb{G}$ is the password dictionary.[14] It then sends $(\mathsf{NewSession}, \mathsf{sid}, \mathsf{client}, \mathsf{server}, \mathsf{pw})$ to client.

2. $\mathcal{Z}$ receives $\mathsf{msg}_1 = (\mathsf{sid}, \mathsf{VK}, A, B, C, D)$ from client (via $\mathcal{A}$) and samples $x_2, y_2, z_2, w_2, r_2 \xleftarrow{\$} \mathbb{Z}_q$. It then sets

$$\begin{aligned}
\alpha' &:= \mathsf{HF}^{\lambda,\Gamma}_{\mathsf{hk}}(\mathsf{sid}, \mathsf{VK}, A, B, C) \\
E &:= g_1^{x_2} g_2^{y_2} h^{z_2} (cd^{\alpha'})^{w_2} \\
F &:= g_1^{r_2} \\
G &:= g_2^{r_2} \\
I &:= h^{r_2} \cdot \mathsf{pw} \\
\beta &:= \mathsf{HF}^{\lambda,\Gamma}_{\mathsf{hk}}(\mathsf{msg}_1, \mathsf{sid}, E, F, G, I) \\
J &:= (cd^{\beta})^{w_2}
\end{aligned}$$

   and instructs $\mathcal{A}$ to send $\mathsf{msg}_2 := (\mathsf{sid}, E, F, G, I, J)$ to client.

3. $\mathcal{Z}$ receives $\mathsf{msg}_3 = (\mathsf{sid}, K, \sigma)$ from client (via $\mathcal{A}$) and $(\mathsf{sid}, \mathsf{sk})$ from client (directly).

4. $\mathcal{Z}$ checks if $\mathsf{Vrfy}_{\mathsf{VK}}(\mathsf{msg}_1, \mathsf{msg}_2, K; \sigma) = 1$. If so, it computes $C' := C/\mathsf{pw}$ and $\mathsf{sk}_S := A^{x_2} B^{y_2} (C')^{z_2} D^{w_2} K^{r_2}$, and outputs 1 if $\mathsf{sk}_S = \mathsf{sk}$. If either of the two checks fails, it outputs 0.

Figure 4: The Un-simulatable Environment $\mathcal{Z}$.

(The logarithm is taken over base $g = g_1$.) We now show that conditioned on $\mathsf{pw} \neq C/A^{\log h}$, the probability that $\mathsf{sk}_S = \mathsf{sk}$ (i.e., $\mathcal{Z}$ outputs 1) is negligible.

Consider the random variables E and sk_S, which are determined completely by the choices of (x_2, y_2, z_2, w_2) and (A, B, C, D, K). We can rewrite these as the following linear system:

$$\begin{pmatrix} \log E \\ \log \mathsf{sk}_S - r_2 \log K \end{pmatrix} = \begin{pmatrix} 1 & \log g_2 & \log h & \log(cd^{\alpha'}) \\ \log A & \log B & \log\left(\frac{C}{\mathsf{pw}}\right) & \log D \end{pmatrix} \begin{pmatrix} x_2 \\ y_2 \\ z_2 \\ w_2 \end{pmatrix}$$

If $C/\mathsf{pw} \neq A^{\log h}$, taking logarithm on both sides we get

$$\log\left(\frac{C}{\mathsf{pw}}\right) \neq \log A \log h,$$

so the two equations in the above linear system are linearly independent (because $(1, \log h)$ and $(\log A, \log(C/\mathsf{pw}))$ are linearly independent). Then for any fixed values of μ, ν, the probability that $\log E = \mu$ and $\log \mathsf{sk}_S - r \log K = \nu$ is exactly $1/q^2$. Hence, the value of sk_S is independent of the value of E, and it follows that the distribution of sk_S is uniform and independent of $\mathcal{S}$'s

[14] Here we assume that the password dictionary $\mathcal{PW}$ is a priori fixed and known to $\mathcal{S}$; the only constraint is that $|\mathcal{PW}| \geq 2$. This only makes the simulation potentially easier. By contrast, in the UC-AGM security proof in Section 4 we do not assume a fixed dictionary. In this way we obtain the strongest possible results in both directions.

view. Recall that sk is client's session key set by $\mathcal{S}$ if $\mathcal{S}$ sent a correct $\mathsf{TestPwd}$ to $\mathcal{F}_{\mathsf{pake}}$, and sampled uniformly at random by $\mathcal{F}_{\mathsf{pake}}$ otherwise. Either way, sk_S is uniform and independent of sk, so the probability that $\mathsf{sk}_S = \mathsf{sk}$ is $1/q$ assuming $C/\mathsf{pw} \neq A^{\log h}$.

Overall, in the ideal world we have that

$$\begin{aligned}\Pr[\mathsf{sk}_S = \mathsf{sk}] &\leq \Pr\left[\frac{C}{\mathsf{pw}} = A^{\log h}\right] + \Pr\left[\mathsf{sk}_S = \mathsf{sk} \wedge \frac{C}{\mathsf{pw}} \neq A^{\log h}\right] \\ &\leq \frac{1}{|\mathcal{PW}|} + \frac{1}{q},\end{aligned}$$

so $\mathcal{Z}$'s distinguishing advantage is at least

$$1 - \frac{1}{|\mathcal{PW}|} - \frac{1}{q},$$

which is non-negligible as long as $|\mathcal{PW}| \geq 2$. This completes the proof. □

Remark 3.1. *We can see that the same proof as above still goes through even if the hash function $\mathsf{HF}^{\lambda,\Gamma}_{\mathsf{hk}}$ is modeled as an RO, i.e., KOY is still not UC-secure even in the ROM. In this regard the security level of KOY is strictly worse than ROM-based PAKE protocols such as EKE and SPAKE2.*

4 UC-Security of KOY in the Algebraic Group Model

In this section, we show the UC-security of the KOY protocol in the Algebraic Group Model (AGM). We adopt the UC-AGM security notion in [ABK+21], which (roughly speaking) requires both the UC adversary and the environment to be algebraic.

Theorem 4.1. *The protocol in Fig. 3 UC-realizes the $\mathcal{F}_{\mathsf{pake}}$ functionality in the Algebraic Group Model under the DSDH assumption for $\mathcal{G}$.*

4.1 Notation

Before we proceed, we describe some notation and terminology. Our proof will proceed via a sequence of games. We will show that the distinguishing advantage of any PPT environment $\mathcal{Z}$ in game $\mathbf{G}_0$ (which corresponds to the real world) and in game $\mathbf{G}_9$ (which corresponds to the ideal world) is negligible. That is, let Z_i denote the event that $\mathcal{Z}$ outputs 1 in game $\mathbf{G}_i$; we need to show that $Z_0 \cong Z_9$.

We refer to the combination of client and server as the *challenger*, which carries out all interactions with the adversary and the environment. In particular, let msg_i $(i = 1, 2, 3)$ denote the i-th message output by the challenger; this message is relayed to the adversary, and the (algebraic) adversary responds with a message msg^*_i (which may or may not be equal to msg_i) along with an algebraic representation "explaining" how the group elements in msg^*_i were computed based on the group elements it has seen so far. client's first message, msg_1, is triggered by the environment's $\mathsf{NewSession}$ message, and when client or server completes a session, the key is output to the environment.

4.2 Proof of Theorem 4.1

We now provide a formal description of the proof of Theorem 4.1. Let $\mathcal{Z}$ be any PPT environment, and $\mathcal{A}$ be the "dummy" adversary that merely passes messages between $\mathcal{Z}$ and protocol parties.

Game $\mathbf{G}_0$: This game corresponds to the real world.

Game $\mathbf{G}'_0$: In this game, we introduce an additional abort clause. If this clause is activated, then the challenger outputs a message Abort and halts.

Let $(\mathsf{VK}, \mathsf{SK})$ be honest outputs of $\mathsf{OTS.KeyGen}(1^\lambda)$ by client. The abort clause is activated if the $(\mathsf{msg}_1^*, \mathsf{msg}_2, \mathsf{msg}_3^*) \neq (\mathsf{msg}_1, \mathsf{msg}_2^*, \mathsf{msg}_3)$ (i.e., $\mathcal{A}$ is not passive), and $\mathsf{Vrfy}_{\mathsf{VK}}(\mathsf{msg}_1^*, \mathsf{msg}_2, K^*; \sigma^*) = 1$ for $\mathsf{msg}_3^* = (K^*, \sigma^*)$.

Note that client's signature is computed as $\sigma \leftarrow \mathsf{Sign}_{\mathsf{SK}}(\mathsf{msg}_1, \mathsf{msg}_2^*, K)$ (where $\mathsf{msg}_3 = (K, \sigma)$). Therefore, the situation amounts to $\mathcal{A}$ coming up with a new message/signature pair $(\mathsf{msg}_1^*, \mathsf{msg}_2, K^*; \sigma^*) \neq (\mathsf{msg}_1, \mathsf{msg}_2^*, K; \sigma)$ such that the verification procedure passes, breaking the strong one-time security of OTS. (This assumes that SK does not repeat across different sessions, which occurs with negligible probability.)

It follows that

$$|\Pr[Z_0] - \Pr[Z'_0]| \leq \mathsf{negl}(\lambda).$$

Game $\mathbf{G}_1$: In this game, the challenger changes the way that crs is sampled. In particular, it uses the initialization algorithm described in Fig. 5. Note that the initialization algorithm closely resembles that of the setup of the Cramer–Shoup encryption scheme.

Initialization: Let $\Gamma[\mathbb{G}, g, q] \leftarrow \mathcal{G}(1^\lambda)$. Set $g_1 := g$ and sample $g_2 \xleftarrow{\$} \bar{\mathbb{G}}$. Sample

$$\kappa \xleftarrow{\$} \mathbb{Z}_q^*; (\chi_1, \chi_2), (\zeta_1, \zeta_2) \xleftarrow{\$} \{(x, y) \in \mathbb{Z}_p^2 \mid g_1^x g_2^y \neq 1\}$$

and set $h := g_1^\kappa$, $c := g_1^{\chi_1} g_2^{\chi_2}$ and $d := g_1^{\zeta_1} g_2^{\zeta_2}$. Furthermore, sample $\mathsf{hk} \xleftarrow{\$} \mathsf{HF.Keyspace}_{\lambda,\Gamma}$. Output $\mathsf{crs} := (\Gamma, g_1, g_2, h, c, d, \mathsf{hk})$.

Figure 5: Challenger's Modified Initialization Algorithm in $\mathbf{G}_1$.

Note that the distribution of crs in $\mathbf{G}_1$ and $\mathbf{G}'_0$ is identical: the only elements which are chosen differently are h, c and d, however their distribution is still uniform in $\bar{\mathbb{G}}$. It follows that

$$\Pr[Z'_0] = \Pr[Z_1].$$

Additional terminology. Before we continue, we introduce some additional, helpful terminology.

- If $\mathsf{msg}_i^* \neq \mathsf{msg}_i$ $(i = 1, 2, 3)$, we say that it is *adversarially generated*. (Intuitively this means that the man-in-the-middle adversary $\mathcal{A}$ modifies msg_i.) Otherwise we say msg_i^* is *honestly generated*.
- Consider adversarially generated $\mathsf{msg}_1^* = (\mathsf{sid}, \mathsf{VK}, A, B, C, D)$. If either $A^{\chi_1 + \alpha\zeta_1} B^{\chi_2 + \alpha\zeta_2} \neq D$ or $C/\mathsf{pw}_{\mathsf{server}} \neq A^\kappa$, then msg_1^* is *invalid*, otherwise it is *valid*. (Intuitively a valid msg_1^* contains the correct password guess for server, and an invalid msg_1^* contains an incorrect password guess or no password guess at all.)

- Consider adversarially generated $\mathsf{msg}_2^* = (\mathsf{sid}, E, F, G, I, J)$. If either $F^{\chi_1+\alpha\zeta_1} G^{\chi_2+\alpha\zeta_2} \neq J$ or $I/\mathsf{pw}_{\mathsf{client}} \neq F^{\kappa}$, then msg_2^* is *invalid*, otherwise it is *valid*. (Intuitively a valid msg_2^* contains the correct password guess for client, and an invalid msg_2^* contains an incorrect password guess or no password guess at all.)

Game $\mathbf{G}_2$: In this game, the challenger makes a change regarding how client's session key $\mathsf{sk}_{\mathsf{client}}$ is computed.

When the challenger receives $\mathsf{msg}_2^* = (\mathsf{sid}, E, F, G, I, J)$, if it is adversarially generated and valid, the challenger considers $\mathcal{A}$'s algebraic representation for E

$$E = g_1^{x_2} g_2^{y_2} h^{z_2} c^{w_2} d^{v_2} A^{x'_2} B^{y'_2} C^{z'_2} D^{w'_2}$$

which was provided along with msg_2^*. It then computes

$$\mathsf{pw} := \frac{I}{F^{\kappa}}, \quad I' := \frac{I}{\mathsf{pw}}, \quad \rho := \frac{C/h^{r_1}}{\mathsf{pw}}$$

and sets the session key $\mathsf{sk}_{\mathsf{client}} := E^{r_1} F^{x_1} G^{y_1} (I')^{z_1} J^{w_1} \rho^{z_2}$. (Among the algebraic coefficients provided by $\mathcal{A}$, only z_2 is used; other coefficients are discarded.) The challenger then responds to the message with msg_3^* exactly as it does in $\mathbf{G}_1$.

Lemma 4.1. $\Pr[Z_1] = \Pr[Z_2]$.

Proof. It is clear that the distribution in $\mathcal{A}$'s (and $\mathcal{Z}$'s) view until an adversarially generated and valid msg_2^* is detected is identical. In the case such a message *is* detected, by the definition of valid messages and the definition of C we have

$$\mathsf{pw} = \frac{I}{F^{\kappa}} = \mathsf{pw}_{\mathsf{client}}, \quad \rho = \frac{C/h^{r_1}}{\mathsf{pw}} = \frac{\mathsf{pw}_{\mathsf{client}}}{\mathsf{pw}} = 1,$$

hence

$$E^{r_1} F^{x_1} G^{y_1} (I')^{z_1} J^{w_1} = E^{r_1} F^{x_1} G^{y_1} (I')^{z_1} J^{w_1} \rho^{z_2}.$$

Thus there is no difference in the distribution of $\mathsf{sk}_{\mathsf{client}}$ between $\mathbf{G}_1$ and $\mathbf{G}_2$. The lemma follows. □

Game $\mathbf{G}_3$: In this game the challenger modifies how it deals with an incoming msg_3^*. Upon receiving this message, the challenger first verifies the signature as in $\mathbf{G}_2$. If the verification passes, it then checks the corresponding msg_1^* of matching sid. If msg_1^* is honestly generated and $\mathsf{pw}_{\mathsf{client}} = \mathsf{pw}_{\mathsf{server}}$, then the challenger finds the unique $\mathsf{sk}_{\mathsf{client}}$ for sid and sets $\mathsf{sk}_{\mathsf{server}} := \mathsf{sk}_{\mathsf{client}}$. In any other case it proceeds as in $\mathbf{G}_2$.

At first glance it might not be clear why such an $\mathsf{sk}_{\mathsf{client}}$ exists. However, this follows from the fact that the game did not Abort. Since msg_1^* is honestly generated, in particular VK in client and server's views is the same, it must be the case that msg_2^* and msg_3^* are also honestly generated (otherwise the game would Abort by the clause introduced in game $\mathbf{G}'_0$). This means that msg_3^* must be sent after client computes msg_3. But client outputs $\mathsf{sk}_{\mathsf{client}}$ at the same time when it computes msg_3, so when the challenger receives msg_3^*, an $\mathsf{sk}_{\mathsf{client}}$ must have already been output.

Furthermore, since $\mathsf{msg}_1^*, \mathsf{msg}_2^*, \mathsf{msg}_3^*$ are all honestly generated (i.e., $\mathcal{A}$ is passive), by the correctness of the protocol, in $\mathbf{G}_2$ it is true that $\mathsf{sk}_{\mathsf{client}} = \mathsf{sk}_{\mathsf{server}}$. Hence, the following unconditionally holds:

$$\Pr[Z_2] = \Pr[Z_3].$$

Game $\mathbf{G}_4$: In this game the challenger deals with an incoming msg_3^* with a corresponding msg_1^* (of the same sid) that is adversarially generated and invalid. If this is the case, it outputs $\mathsf{sk}_{\mathsf{server}} \xleftarrow{\$} \mathbb{G}$. In any other case it proceeds as in $\mathbf{G}_3$.

Lemma 4.2. $\Pr[Z_3] = \Pr[Z_4]$.

Proof. We will proceed via an information-theoretic argument. Since msg_1^* is adversarially generated and invalid, there are two possibilities: either $A^{\chi_1+\alpha\zeta_1}B^{\chi_2+\alpha\zeta_2} \neq D$, or $C/\mathsf{pw}_{\mathsf{server}} \neq A^{\kappa}$. Consider now the random variables E and $\mathsf{sk}_{\mathsf{server}}$, which are determined completely by the choices of (x_2, y_2, z_2, w_2) and (A, B, C, D, K). We can rewrite these as the following linear system, in which the logarithms are taken over base $g = g_1$:

$$\begin{pmatrix} \log E \\ \log \mathsf{sk}_{\mathsf{server}} - r_2 \log K \end{pmatrix} = \begin{pmatrix} 1 & \log g_2 & \kappa & \log(cd^{\alpha}) \\ \log A & \log B & \log\left(\frac{C}{\mathsf{pw}_{\mathsf{server}}}\right) & \log D \end{pmatrix} \begin{pmatrix} x_2 \\ y_2 \\ z_2 \\ w_2 \end{pmatrix}. \tag{1}$$

We claim that the two equations in the above linear system are linearly independent:

- If $A^{\chi_1+\alpha\zeta_1}B^{\chi_2+\alpha\zeta_2} \neq D$, then taking logarithm on both sides we get

  $$(\chi_1 + \alpha\zeta_1)\log A + (\chi_2 + \alpha\zeta_2)\log B \neq \log D.$$

  However, note that $cd^{\alpha} = (g_1^{\chi_1}g_2^{\chi_2})(g_1^{\zeta_1}g_2^{\zeta_2})^{\alpha} = g_1^{\chi_1+\alpha\zeta_1}g_2^{\chi_2+\alpha\zeta_2}$, so

  $$(\chi_1 + \alpha\zeta_1) + (\chi_2 + \alpha\zeta_2)\log g_2 = \log(cd^{\alpha}),$$

  so the two equations in the above linear system are linearly independent (because $(1, \log g_2, \log(cd^{\alpha}))$ and $(\log A, \log B, \log D)$ are linearly independent).

- If $C/\mathsf{pw}_{\mathsf{server}} \neq A^{\kappa}$, then taking logarithm on both sides we get

  $$\log\left(\frac{C}{\mathsf{pw}_{\mathsf{server}}}\right) \neq \kappa \log A,$$

  so the two equations in the above linear system are linearly independent (because $(1, \kappa)$ and $(\log A, \log(C/\mathsf{pw}_{\mathsf{server}}))$ are linearly independent).

Then for any fixed values of μ, ν, the probability that $\log E = \mu$ and $\log \mathsf{sk}_{\mathsf{server}} - r \log K = \nu$ is exactly $1/q^2$. Hence, the value of $\mathsf{sk}_{\mathsf{server}}$ is independent of the value of E, and it follows that the distribution of $\mathsf{sk}_{\mathsf{server}}$ is uniform and independent of $\mathcal{Z}$'s view in $\mathbf{G}_3$, which is the same as $\mathbf{G}_4$. □

Game $\mathbf{G}_5$: In this game the challenger deals with an incoming msg_3^* with a corresponding msg_1^* (of the same sid) that is honestly generated. If this is the case and $\mathsf{pw}_{\mathsf{client}} \neq \mathsf{pw}_{\mathsf{server}}$, it outputs $\mathsf{sk}_{\mathsf{server}} \xleftarrow{\$} \mathbb{G}$. In any other case it proceeds as in $\mathbf{G}_4$.

Lemma 4.3. $\Pr[Z_4] = \Pr[Z_5]$.

Proof. Since msg_1^* is honestly generated, $\mathsf{msg}_1^* = \mathsf{msg}_1$. By the definition of msg_1, $C/\mathsf{pw}_{\mathsf{client}} = A^{\kappa}$, and since $\mathsf{pw}_{\mathsf{client}} \neq \mathsf{pw}_{\mathsf{server}}$, we have $C/\mathsf{pw}_{\mathsf{server}} \neq A^{\kappa}$. The rest of the argument proceeds exactly as in the second bullet of Lemma 4.2. □

Recap. Before we proceed, we review the changes we have made in preceding games. Upon receiving msg_3^*, if the signature verification passes, the challenger checks the corresponding msg_1^* of matching sid and determines server's session key $\mathsf{sk}_{\mathsf{server}}$ using the following simplified procedure:

- If msg_1^* is honestly generated and $\mathsf{pw}_{\mathsf{client}} = \mathsf{pw}_{\mathsf{server}}$, then the challenger finds the unique $\mathsf{sk}_{\mathsf{client}}$ for sid and sets $\mathsf{sk}_{\mathsf{server}} := \mathsf{sk}_{\mathsf{client}}$ (changed in $\mathbf{G}_3$).
- If msg_1^* is honestly generated and $\mathsf{pw}_{\mathsf{client}} \neq \mathsf{pw}_{\mathsf{server}}$, then the challenger samples $\mathsf{sk}_{\mathsf{server}} \xleftarrow{\$} \mathbb{G}$ (changed in $\mathbf{G}_5$).
- If msg_1^* is adversarially generated and invalid, then the challenger samples $\mathsf{sk}_{\mathsf{server}} \xleftarrow{\$} \mathbb{G}$ (changed in $\mathbf{G}_4$).

Finally, if msg_1^* is adversarially generated and valid, we have not made any changes yet and the challenger computes $\mathsf{sk}_{\mathsf{server}}$ as in the real world.

Game $\mathbf{G}_6$: In this game the challenger modifies how it determines msg_2. On receiving msg_1^*, if it is adversarially generated and valid, the challenger proceeds as in $\mathbf{G}_5$. Otherwise, instead of setting $I := h^{r_2} \cdot \mathsf{pw}_{\mathsf{server}}$, it sets $I := h^{r_2} \cdot g^{N+1}$ where g^{N+1} is a "dummy" plaintext that does not belong to the password dictionary $\mathcal{PW}$ (so it is guaranteed that $g^{N+1} \neq \mathsf{pw}_{\mathsf{server}}$).[15]

Lemma 4.4. *Under the DDH assumption for $\mathcal{G}$, $|\Pr[Z_5] - \Pr[Z_6]| \leq \mathsf{negl}(\lambda)$.*

Proof. $\mathcal{Z}$'s views in $\mathbf{G}_5$ and $\mathbf{G}_6$ are identical if msg_1^* is adversarially generated and valid. Below we assume that this does occur.

By Section 2.3, under DDH the Cramer–Shoup with labels encryption scheme (Fig. 1) satisfies CCA-security. We show that any environment $\mathcal{Z}$ that can distinguish between $\mathbf{G}_5$ and $\mathbf{G}_6$, can be used as a subroutine to build a reduction $\mathcal{R}_1$ that can break the CCA-security for Cramer–Shoup with labels.

$\mathcal{R}_1$ begins by receiving the Cramer–Shoup public key $(\Gamma, \mathsf{hk}, g_1, g_2, c, d, h)$. It sets $\mathsf{crs} := (\Gamma, g_1, g_2, h, c, d, \mathsf{hk})$ (instead of running the initialization procedure in Fig. 5) and begins a simulation of the challenger for $\mathcal{Z}$. This simulation goes as follows:

- On receiving a message $\mathsf{msg}_1^* = (\mathsf{sid}, \mathsf{VK}, A, B, C, D)$, if it is adversarially generated, $\mathcal{R}_1$ needs to determine whether this message is valid or not. Since $\mathcal{R}_1$ does not hold the Cramer–Shoup secret key, it cannot check it directly. Instead, $\mathcal{R}_1$ queries the decryption oracle on $((\mathsf{sid}, \mathsf{VK}); (A, B, C, D))$, and sets msg_1^* to be valid if the result is $\mathsf{pw}_{\mathsf{server}}$ and invalid otherwise. ($\mathsf{pw}_{\mathsf{server}}$ was provided by $\mathcal{Z}$ while initiating client's session.) If msg_1^* is valid, $\mathcal{R}_1$ outputs 1 and halts. Otherwise (i.e., if msg_1^* is invalid, or honestly generated) $\mathcal{R}_1$ submits $((\mathsf{sid}, \mathsf{VK}); m_0 := \mathsf{pw}_{\mathsf{server}})$ and $((\mathsf{sid}, \mathsf{VK}); m_1 := g^{N+1})$ as label/plaintext pairs for the encryption oracle. It receives in return the target ciphertext ψ^*, and includes $(F, G, I, J) := \psi^*$ as part of msg_2.
- On receiving a message msg_2^*, $\mathcal{R}_1$ determines $\mathsf{sk}_{\mathsf{client}}$ as in $\mathbf{G}_5$. This can be done because we have not made any change on the client's side, so $\mathcal{R}_1$ can simulate it exactly as what client does (unlike on the server's side where (F, G, I, J) is generated by the encryption oracle and $\mathcal{R}_1$ does not know the randomness r_2 used therein).

[15] This assumes that $\mathcal{PW} = \{g, \ldots, g^N\}$ is public. If it is not, the challenger can instead sample a "dummy" plaintext $\widetilde{\mathsf{pw}}$ uniformly at random from $\mathbb{G}$, and $\widetilde{\mathsf{pw}} \neq \mathsf{pw}_{\mathsf{server}}$ holds with overwhelming probability. The same goes for game $\mathbf{G}_9$ and the simulator below.

- On receiving a message msg_3^*, $\mathcal{R}_1$ first verifies the signature. If the verification passes, it checks the corresponding msg_1^* of matching sid; this message must be either honestly generated, or adversarially generated and invalid (if it were adversarially generated and valid, $\mathcal{R}_1$ would have already halted in the first step). Either way, $\mathcal{R}_1$ can determine $\mathsf{sk}_{\mathsf{server}}$ based on the three cases listed in the "recap" paragraph above.

Finally, $\mathcal{R}_1$ outputs what $\mathcal{Z}$ outputs.

$\mathcal{R}_1$ does not make any illegal decryption oracle queries, since in the current session it only makes a single decryption oracle query (if any), which is *before* receiving the target ciphertext. $\mathcal{R}_1$ may make additional decryption oracle queries in other sessions, but they have different labels than the target ciphertext (since sid is included in the label) and thus cannot be illegal.

It is clear that $\mathcal{R}_1$ simulates $\mathbf{G}_5$ if the returned ciphertext ψ^* is an encryption of $\mathsf{pw}_{\mathsf{server}}$, and $\mathbf{G}_6$ if ψ^* is an encryption of g^{N+1}. It follows that

$$|\Pr[Z_5] - \Pr[Z_6]| \leq \mathsf{AdvCCA}_{\mathcal{R}_1}(\lambda) \leq \mathsf{negl}(\lambda). \qquad \square$$

Game $\mathbf{G}_7$: In this game the challenger deals with incoming message msg_2^*. If msg_2^* is adversarially generated and invalid, then the challenger outputs $\mathsf{sk}_{\mathsf{client}} \xleftarrow{\$} \mathbb{G}$.

The proof of indistinguishability of this game is very similar to that of game $\mathbf{G}_4$. Here either $F^{\chi_1+\beta\zeta_1}G^{\chi_2+\beta\zeta_2} \neq J$ or $I/\mathsf{pw}_{\mathsf{client}} \neq F^{\kappa}$. A linear system similar to Eq. (1) follows, and the proof of linear independence is also nearly identical. We can conclude that $\mathsf{sk}_{\mathsf{client}}$ is uniformly random in $\mathbf{G}_6$, and equivalence follows immediately:

$$\Pr[Z_6] = \Pr[Z_7].$$

Game $\mathbf{G}_8$: In this game the challenger again deals with incoming message msg_2^*. If msg_2^* is honestly generated, then the challenger outputs $\mathsf{sk}_{\mathsf{client}} \xleftarrow{\$} \mathbb{G}$.

Lemma 4.5. $\Pr[Z_7] = \Pr[Z_8]$.

Proof. Note that in $\mathbf{G}_6$ the challenger computes I as $h^{r_2} \cdot g^{N+1} = g^{r_2\kappa} \cdot g^{N+1} = F^{\kappa} \cdot g^{N+1}$, so if msg_2^* is honestly generated, it is guaranteed that $I/\mathsf{pw}_{\mathsf{client}} \neq F^{\kappa}$. Then the argument is identical to $\mathbf{G}_7$. $\square$

Game $\mathbf{G}_9$: In this game the challenger modifies how it determines msg_1. On receiving an opening $(\mathsf{NewSession}, \mathsf{sid}, \mathsf{client}, \mathsf{server}, \mathsf{pw}_{\mathsf{client}}, \text{"client"})$ message, instead of setting $C := h^{r_1} \cdot \mathsf{pw}_{\mathsf{client}}$, the challenger sets $C := h^{r_1} \cdot g_1^{N+1}$ where g^{N+1} is a "dummy" plaintext that does not belong to the password dictionary $\mathcal{PW}$.

Lemma 4.6. *Under the DSDH assumption for $\mathcal{G}$,* $|\Pr[Z_8] - \Pr[Z_9]| \leq \mathsf{negl}(\lambda)$.

Proof. By Theorem A.1, under DSDH the Cramer–Shoup with labels encryption scheme (Fig. 1) satisfies DR-CCA-security with auxiliary information $\mathsf{aux} = (\chi_1, \chi_2, \zeta_1, \zeta_2, g_1^{r_1^2}, g_2^{r_1^2}, (h^{r_1} \cdot m_b)^{r_1}, (cd^{\alpha})^{r_1^2})$. We show that any environment $\mathcal{Z}$ that can distinguish between $\mathbf{G}_8$ and $\mathbf{G}_9$, can be used as a subroutine to build a reduction $\mathcal{R}_2$ that can break the DR-CCA-security for Cramer–Shoup with labels.

$\mathcal{R}_2$ begins by receiving the Cramer–Shoup public key $(\Gamma, \mathsf{hk}, g_1, g_2, c, d, h)$. It sets $\mathsf{crs} := (\Gamma, g_1, g_2, h, c, d, \mathsf{hk})$ (instead of running the initialization procedure in Fig. 5) and begins a simulation of the challenger for $\mathcal{Z}$. This simulation goes as follows:

- On receiving $(\mathsf{NewSession}, \mathsf{sid}, \mathsf{client}, \mathsf{server}, \mathsf{pw}_{\mathsf{client}}, \text{“client”})$, $\mathcal{R}_2$ runs $(\mathsf{VK}, \mathsf{SK}) \leftarrow \mathsf{OTS.Gen}(1^\lambda)$ and submits $((\mathsf{sid}, \mathsf{VK}); m_0 := \mathsf{pw}_{\mathsf{client}})$ and $((\mathsf{sid}, \mathsf{VK}); m_1 := g^{N+1})$ as label/plaintext pairs for the encryption oracle. It receives in return the target ciphertext ψ^*, and includes $(A, B, C, D) := \psi^*$ as part of msg_1.

- On receiving a message msg_1^*, if it is adversarially generated, $\mathcal{R}_2$ determines whether this message is valid or not as follows. Since $\mathcal{R}_2$ does not hold the Cramer–Shoup secret key, it cannot check it directly. Instead, $\mathcal{R}_2$ queries the decryption oracle on $((\mathsf{sid}, \mathsf{VK}); (A, B, C, D))$, and sets msg_1^* to be valid if the result is $\mathsf{pw}_{\mathsf{server}}$ and invalid otherwise. After that, $\mathcal{R}_2$ generates msg_2 as in $\mathbf{G}_8$.

- On receiving a message $\mathsf{msg}_2^* = (\mathsf{sid}, E, F, G, I, J)$, if it is adversarially generated, $\mathcal{R}_2$ again needs to determine whether this message is valid or not. $\mathcal{R}_2$ does this by querying the decryption oracle on $((\mathsf{msg}_1^*, E); (F, G, I, J))$, and sets msg_2^* to be valid if the result is $\mathsf{pw}_{\mathsf{client}}$ and invalid otherwise.

- After that, $\mathcal{R}_2$ determines $\mathsf{sk}_{\mathsf{client}}$ as follows. If msg_2^* is either honestly generated, or adversarially generated and invalid, $\mathcal{R}_2$ samples $\mathsf{sk}_{\mathsf{client}} \xleftarrow{\$} \mathbb{G}$ (this matches $\mathbf{G}_7$ and $\mathbf{G}_8$).
  The more difficult case is when msg_2^* is adversarially generated and valid. Consider $\mathcal{A}$'s algebraic representation for E

$$E = g_1^{x_2} g_2^{y_2} h^{z_2} c^{w_2} d^{v_2} A^{x'_2} B^{y'_2} C^{z'_2} D^{w'_2}$$

  which was provided along with msg_2^*, and recall that ρ is defined in $\mathbf{G}_2$ as

$$\rho = \frac{C/h^{r_1}}{I/F^\kappa} = \frac{C/h^{r_1}}{\mathsf{pw}_{\mathsf{client}}}$$

  (where the second equation holds since msg_2^* is valid). Let $X_1 = E^{r_1} \rho^{z_2}$; we first show how $\mathcal{R}_2$ can compute X_1 in such a way that matches $\mathbf{G}_8$ if the target ciphertext ψ^* is an encryption of $\mathsf{pw}_{\mathsf{client}}$ and $\mathbf{G}_9$ if ψ^* is an encryption of g^{N+1}. We have

$$E^{r_1} = g_1^{r_1 x_2} g_2^{r_1 y_2} h^{r_1 z_2} c^{r_1 w_2} d^{r_1 v_2} A^{r_1 x'_2} B^{r_1 y'_2} C^{r_1 z'_2} D^{r_1 w'_2},$$
$$\rho^{z_2} = \left(\frac{C}{\mathsf{pw}_{\mathsf{client}}}\right)^{z_2} \cdot \frac{1}{h^{r_1 z_2}},$$

  so

$$X_1 = g_1^{r_1 x_2} g_2^{r_1 y_2} \left(\frac{C}{\mathsf{pw}_{\mathsf{client}}}\right)^{z_2} c^{r_1 w_2} d^{r_1 v_2} A^{r_1 x'_2} B^{r_1 y'_2} C^{r_1 z'_2} D^{r_1 w'_2}$$

  in both $\mathbf{G}_8$ and $\mathbf{G}_9$. $\mathcal{R}_2$ can compute X_1 according to the last formula because:

  - $\mathcal{R}_2$ knows $x_2, y_2, z_2, w_2, v_2, x'_2, y'_2, z'_2, w'_2$ from $\mathcal{A}$'s algebraic representation of E;
  - $\mathcal{R}_2$ knows $g_1^{r_1} = A, g_2^{r_1} = B$ from its target ciphertext ψ^*;
  - $\mathcal{R}_2$ knows C from ψ^* and $\mathsf{pw}_{\mathsf{client}}$ from $\mathcal{Z}$'s $\mathsf{NewSession}$ message; and

– $\mathcal{R}_2$ knows

$$
\begin{aligned}
c^{r_1} &= (g_1^{\chi_1} g_2^{\chi_2})^{r_1} = (g_1^{r_1})^{\chi_1} (g_2^{r_1})^{\chi_2} = A^{\chi_1} B^{\chi_2} \\
d^{r_1} &= (g_1^{\zeta_1} g_2^{\zeta_2})^{r_1} = (g_1^{r_1})^{\zeta_1} (g_2^{r_1})^{\zeta_2} = A^{\zeta_1} B^{\zeta_2} \\
A^{r_1} &= g_1^{r_1^2} \\
B^{r_1} &= g_2^{r_1^2} \\
C^{r_1} &= (h^{r_1} \cdot m_b)^{r_1} \\
D^{r_1} &= (cd^{\alpha})^{r_1^2}
\end{aligned}
$$

by querying the auxiliary oracle.

After computing X_1, $\mathcal{R}_2$ samples $x_1, y_1, z_1, w_1 \xleftarrow{\$} \mathbb{Z}_q$ and computes $X_2 = F^{x_1} G^{y_1} (I/\mathsf{pw}_{\mathsf{client}})^{z_1} J^{w_1}$. This gives $\mathsf{sk}_{\mathsf{client}} = X_1 X_2$ which matches $\mathbf{G}_8$ if ψ^* is an encryption of $\mathsf{pw}_{\mathsf{client}}$ and $\mathbf{G}_9$ if ψ^* is an encryption of g^{N+1}.

- $\mathcal{R}_2$ generates msg_3 as in $\mathbf{G}_8$ (if msg_2^* is adversarially generated and valid, $\mathcal{R}$ uses the same x_1, y_1, z_1, w_1 while computing $\mathsf{sk}_{\mathsf{client}}$).
- On receiving a message msg_3^*, $\mathcal{R}_2$ determines $\mathsf{sk}_{\mathsf{server}}$ as in $\mathbf{G}_8$. (If this message is adversarially generated, this requires determining whether it is valid or not, which has already been done in the second bullet above.)

Finally, $\mathcal{R}_2$ outputs what $\mathcal{Z}$ outputs.

We argue that $\mathcal{R}_2$ does not make any illegal decryption oracle queries. In the current session it makes up to two decryption oracle queries: on $((\mathsf{sid}, \mathsf{VK}); (A, B, C, D))$ and on $((\mathsf{msg}_1^*, E); (F, G, I, J))$. The former query is made only if $\mathsf{msg}_1^* = (\mathsf{sid}, \mathsf{VK}, A, B, C, D)$ is adversarially generated, so at least one of the label and the ciphertext must be different from the target ciphertext. The latter query is legal since its label has a different format than the target ciphertext. $\mathcal{R}_2$ may make additional decryption oracle queries in other sessions, but they have different labels than the target ciphertext (since sid is included in the label) and thus cannot be illegal.

From the analysis above, it is clear that $\mathcal{R}_2$ simulates $\mathbf{G}_8$ if the returned ciphertext ψ^* is an encryption of $\mathsf{pw}_{\mathsf{client}}$, and $\mathbf{G}_9$ if ψ^* is an encryption of g^{N+1}. It follows that

$$|\Pr[Z_8] - \Pr[Z_9]| \leq \mathsf{AdvDRCCA}_{\mathcal{R}_2}(\lambda) \leq \mathsf{negl}(\lambda). \qquad \square$$

Remark 4.1. *It is interesting to compare game* $\mathbf{G}_6$ *and* $\mathbf{G}_9$*: they respectively change* msg_2 *and* msg_1 *to encryptions of the "dummy" plaintext, yet* $\mathbf{G}_6$ *can be reduced to the CCA-security of Cramer–Shoup (in which DDH suffices), while* $\mathbf{G}_9$ *can only be reduced to the DR-CCA-security of Cramer–Shoup (which requires DSDH). The critical difference lies in the case that the adversarial message (*msg_1^* *in* $\mathbf{G}_6$*,* msg_2^* *in* $\mathbf{G}_9$*) is adversarially generated and valid:*

- *In* $\mathbf{G}_6$ *the adversarial message* msg_1^* *is sent* before msg_2 *is generated, so the challenger is able to generate* msg_2 *according to whether* msg_1^* *is valid or not. If* msg_1^* *is adversarially generated and valid, the challenger can choose not to make any change*[16]*, allowing the reduction to only deal with other cases — where* $\mathsf{sk}_{\mathsf{server}}$ *can be computed normally.*

[16] Alternatively, the challenger could always encrypt the "dummy" plaintext in msg_2 regardless of whether msg_1^* is valid or not. The argument would still go through, yet it would require a reduction to the DR-CCA-security of Cramer–Shoup — similar to $\mathbf{G}_9$.

- *In contrast, in* $\mathbf{G}_9$ *the adversarial message* msg_2^* *is sent* after msg_1 *is generated, so the challenger has to generate* msg_1 *as an encryption of the "dummy" plaintext regardless of whether* msg_2^* *is valid or not. This means that the reduction has to compute* $\mathsf{sk}_{\mathsf{client}}$ *even if* msg_2^* *is adversarially generated and valid, which cannot be done normally exactly because* msg_1 *is not an encryption of the correct password* $\mathsf{pw}_{\mathsf{client}}$.

Also note that this distinction is non-existent in the game-based security proof in [KOY01]: there, once the challenger receives an adversarially generated and valid message, both games (and both reductions) can halt and let the adversary win (by outputting a "dummy" symbol ∇*), and there is no need to compute the session key. Since the DR-CCA-security of Cramer–Shoup only appears in the computation of* $\mathsf{pw}_{\mathsf{client}}$ *in* $\mathbf{G}_9$*, the game-based security of the KOY protocol does not need it and can be based on DDH.*

The ideal world: We now introduce the simulator $\mathcal{S}$ shown in Fig. 6. With a tedious but straightforward check, we can see that $\mathcal{S}$ does exactly what is done by the challenger of $\mathbf{G}_9$, except for determining whether a message is invalid or valid, which is done by $\mathcal{F}_{\mathsf{pake}}$. Once this has been determined, it continues the simulation and finds the session key $\mathsf{sk}_{\mathsf{client}}$ or $\mathsf{sk}_{\mathsf{server}}$, which is delegated to $\mathcal{F}_{\mathsf{pake}}$.

Remark 4.2. *Similar to Remark 4.1, there is a critical difference between how* $\mathsf{sk}_{\mathsf{client}}$ *is computed (which follows game* $\mathbf{G}_2$*) and how* $\mathsf{sk}_{\mathsf{server}}$ *is computed:*

- msg_1 *is changed to an encryption of the "dummy" plaintext not equal to* $\mathsf{pw}_{\mathsf{client}}$*, so while computing* $\mathsf{sk}_{\mathsf{client}}$ *the simulator has to "correct its previous mistake" using the factor* ρ *(whose computation requires the AGM) — which is what we do in* $\mathbf{G}_2$.
- *However,* msg_2 *remains an encryption of the "real"* $\mathsf{pw}_{\mathsf{server}}$ *if* msg_1^* *is adversarially generated and valid, so in this case the simulator does not make any "mistake" to begin with while generating* msg_2*, and* $\mathsf{sk}_{\mathsf{server}}$ *can be computed normally.*

From the preceding games we conclude that under the DSDH assumption for $\mathcal{G}$,

$$|\Pr[Z_0] - \Pr[Z_9]| \leq \mathsf{negl}(\lambda),$$

which completes the proof. □

5 Discussion

There are two ways to interpret our contributions. One is to view them as results on PAKE: We study the security of the seminal KOY protocol — one of the first PAKE protocols with a security proof — in the UC setting, and give both negative and positive results. It is interesting to review the timeline here:

- In 2001 the KOY protocol was published [KOY01], with a game-based security proof under the DDH assumption;
- In 2005 the UC security notion for PAKE was proposed [CHK+05], and over the years it has largely superseded the game-based security notion and become the standard. The UC-secure PAKE protocol in [CHK+05] is based on (a generalization of) KOY, but adds a NIZK proof which destroys the efficiency of the protocol;

Public parameters:
$\mathcal{S}$ generates crs as described in Fig. 5. This in particular allows $\mathcal{S}$ to record $\kappa, \chi_1, \chi_2, \zeta_1, \zeta_2$.

On receiving $(\mathsf{NewSession}, \mathsf{sid}, \mathsf{P}, \mathsf{P}', \mathbf{role})$ **from** $\mathcal{F}_{\mathsf{pake}}$:

If $\mathsf{P} = \mathsf{client}$:

- **In Round $\mathbf{R}_1$:** $\mathcal{S}$ samples $(\mathsf{VK}, \mathsf{SK}) \leftarrow \mathsf{OTS.Gen}(1^\lambda)$ and $r_1 \xleftarrow{\$} \mathbb{Z}_q$. It sets
$$\begin{aligned} A &:= g_1^{r_1} \\ B &:= g_2^{r_1} \\ C &:= h^{r_1} \cdot g^{N+1} \\ \alpha &:= \mathsf{HF}_{\mathsf{hk}}^{\lambda,\Gamma}(\mathsf{sid}, \mathsf{VK}, A, B, C) \\ D &:= (cd^\alpha)^{r_1} \end{aligned}$$
and sends $\mathsf{msg}_1 = (\mathsf{sid}, \mathsf{VK}, A, B, C, D)$.
- **On receiving** $\mathsf{msg}_2^* = (\mathsf{sid}, E, F, G, I, J)$**:** If $\mathsf{msg}_2^* \neq \mathsf{msg}_2$, $\mathcal{S}$ first checks if $F^{\chi_1+\alpha\zeta_1} G^{\chi_2+\alpha\zeta_2} = J$. If not, $\mathcal{S}$ submits $(\mathsf{TestPwd}, \mathsf{sid}, \mathsf{client}, \perp)$ to $\mathcal{F}_{\mathsf{pake}}$. Else $\mathcal{S}$ submits $(\mathsf{TestPwd}, \mathsf{sid}, \mathsf{client}, I/F^\kappa)$ to $\mathcal{F}_{\mathsf{pake}}$.
  - **If $\mathcal{F}_{\mathsf{pake}}$ outputs "correct guess":** $\mathcal{S}$ continues the simulation as described in $\mathbf{G}_2$ and submits $(\mathsf{NewKey}, \mathsf{sid}, \mathsf{client}, \mathsf{sk}_{\mathsf{client}})$ to $\mathcal{F}_{\mathsf{pake}}$.
  - **If $\mathcal{F}_{\mathsf{pake}}$ outputs "wrong guess":** $\mathcal{S}$ continues the simulation as described in $\mathbf{G}_7$ and submits $(\mathsf{NewKey}, \mathsf{sid}, \mathsf{client}, 1_{\mathbb{G}})$ to $\mathcal{F}_{\mathsf{pake}}$.

  If $\mathsf{msg}_2^* = \mathsf{msg}_2$, $\mathcal{S}$ skips $\mathsf{TestPwd}$, continues the simulation as described in $\mathbf{G}_8$, and submits $(\mathsf{NewKey}, \mathsf{sid}, \mathsf{client}, 1_{\mathbb{G}})$ to $\mathcal{F}_{\mathsf{pake}}$.
- **In Round $\mathbf{R}_3$ (Client):** $\mathcal{S}$ samples $x_1, y_1, z_1, w_1 \xleftarrow{\$} \mathbb{Z}_q$. It finds $\mathsf{msg}_1, \mathsf{msg}_2^*$ of the same sid and sets
$$\begin{aligned} \beta' &:= \mathsf{HF}_{\mathsf{hk}}^{\lambda,\Gamma}(\mathsf{msg}_1, \mathsf{sid}, E, F, G, I) \\ K &:= g_1^{x_1} g_2^{x_2} h^{z_1} (cd^{\beta'})^{w_1} \\ \sigma &\leftarrow \mathsf{Sign}_{\mathsf{SK}}(\mathsf{msg}_1, \mathsf{msg}_2^*, K) \end{aligned}$$
and sends $\mathsf{msg}_3 = (\mathsf{sid}, K, \sigma)$.

If $\mathsf{P} = \mathsf{server}$**:**

- **On receiving** $\mathsf{msg}_1^* = (\mathsf{sid}, \mathsf{VK}, A, B, C, D)$**:** If $\mathsf{msg}_1^* \neq \mathsf{msg}_1$, $\mathcal{S}$ first checks if $A^{\chi_1+\alpha\zeta_1} B^{\chi_2+\alpha\zeta_2} = D$. If not, it submits $(\mathsf{TestPwd}, \mathsf{sid}, \mathsf{client}, \perp)$ to $\mathcal{F}_{\mathsf{pake}}$. Else it submits $(\mathsf{TestPwd}, \mathsf{sid}, \mathsf{client}, C/A^\kappa)$ to $\mathcal{F}_{\mathsf{pake}}$. $\mathcal{S}$ records $\mathcal{F}_{\mathsf{pake}}$'s response "correct/wrong guess".
  If $\mathsf{msg}_1^* = \mathsf{msg}_1$, $\mathcal{S}$ skips $\mathsf{TestPwd}$.
  If there was a $\mathsf{TestPwd}$ message to $\mathcal{F}_{\mathsf{pake}}$ resulting in "correct guess", $\mathcal{S}$ sets $\widetilde{\mathsf{pw}} := C/A^\kappa$. Else it sets $\widetilde{\mathsf{pw}} := g^{N+1}$.
- **In Round $\mathbf{R}_2$:** $\mathcal{S}$ samples $x_2, y_2, z_2, w_2, r_2 \xleftarrow{\$} \mathbb{Z}_q$. It sets
$$\begin{aligned} \alpha' &:= \mathsf{HF}_{\mathsf{hk}}^{\lambda,\Gamma}(\mathsf{sid}, \mathsf{VK}, A, B, C) \\ E &:= g_1^{x_2} g_2^{y_2} h^{z_2} (cd^{\alpha'})^{w_2} \\ F &:= g_1^{r_2} \\ G &:= g_2^{r_2} \\ I &:= h^{r_2} \cdot \widetilde{\mathsf{pw}} \\ \beta &:= \mathsf{HF}_{\mathsf{hk}}^{\lambda,\Gamma}(\mathsf{msg}_1^*, \mathsf{sid}, E, F, G, I) \\ J &:= (cd^\beta)^{r_2} \end{aligned}$$
and sends $\mathsf{msg}_2 = (\mathsf{sid}, E, F, G, I, J)$.
- **On receiving** $\mathsf{msg}_3^* = (\mathsf{sid}, K, \sigma)$**:** $\mathcal{S}$ finds $\mathsf{msg}_1^*, \mathsf{msg}_2$ of the same sid and checks if $\mathsf{Vrfy}_{\mathsf{VK}}(\mathsf{msg}_1^*, \mathsf{msg}_2, K; \sigma) = 1$. If not, $\mathcal{S}$ submits $(\mathsf{Abort}, \mathsf{sid}, \mathsf{server})$ to $\mathcal{F}_{\mathsf{pake}}$.
  Else, $\mathcal{S}$ checks if there was a $\mathsf{TestPwd}$ message to $\mathcal{F}_{\mathsf{pake}}$ resulting in "correct guess" on receiving msg_1^*. If so, it submits $(\mathsf{NewKey}, \mathsf{sid}, \mathsf{client}, \mathsf{sk}_{\mathsf{server}} := A^{x_2} B^{y_2} (C/\widetilde{\mathsf{pw}})^{z_2} D^{w_2} K^{r_2})$ to $\mathcal{F}_{\mathsf{pake}}$. Else, it submits $(\mathsf{NewKey}, \mathsf{sid}, \mathsf{client}, 1_{\mathbb{G}})$ to $\mathcal{F}_{\mathsf{pake}}$.

Figure 6: The Simulator for Theorem 4.1.

- In 2018 the AGM was (formally) introduced [FKL18]. Its original intention was to reanalyze some group-based assumptions (e.g., the equivalence between DL and CDH) and game-based security of some schemes (e.g., tight security of the BLS signature scheme) under a restricted class of adversaries, without UC in mind;
- In 2021 people realized that the AGM could be imposed in the UC framework as well [ABK+21], defining the UC-AGM framework in which some RO-based PAKE protocols such as SPAKE2 and CPace can be proven secure;
- Finally, this work revisits the KOY protocol and proves that (1) it is unconditionally *not* UC-secure, and (2) it is UC-AGM-secure, but under the DSDH assumption that is stronger than DDH under which KOY is game-based secure. This yields the first UC-AGM-security analysis of a PAKE protocol without random oracles.

KOY is not the most efficient non-ROM PAKE protocol; subsequent works such as [JG04] (JG) reuse the "encrypt then SPHF" idea of KOY but make improvements on both computational cost and round complexity. In the meantime, they also seem to inherit the UC-insecurity of KOY, due to essentially the same reason. It is hopeful that one could also salvage the UC-security of the JG protocol in the AGM.

On the other hand, our work could also be viewed as a discussion of the gaps between game-based and simulation-based security notions, using PAKE (and the KOY protocol in particular) as a case study. The central question here is

There are many cases where a protocol is game-based secure but not simulation-based secure. Why is that?

There has been surprisingly little formal study on this, but two reasons appear well-known as "folklore":

1. A simulator needs to extract a corrupt party's input (or the man-in-the-middle adversary's input in PAKE), which it sometimes cannot do; and
2. A simulator needs to compute an honest party's output, which it sometimes cannot do.[17]

(1) is not an issue for the KOY protocol, but the UC-security of KOY fails precisely due to (2). Our first takeaway is that

> *The simulator might be able to complete its (otherwise impossible) task in some idealized models that force the adversary to output certain additional information.*

In our specific example of KOY, the idealized model is the AGM, but we feel the above is a general principle that might inspire the introduction of new idealized models in other (non-DH-group) settings.

[17] In many MPC settings where the functionality merely evaluates a function and sends the results to corresponding parties, (2) is not a concern: after extracting a corrupt party's input, the simulator can simply feed it to the functionality, who will compute the honest party's output for the simulator. But this is not necessarily the case for more complicated functionalities such as PAKE.

However, our work reveals yet a third reason for which simulation might fail, which seems to have received relatively little attention (and certainly has not been studied in the context of PAKE):

> *Simulation may fail because after the adversary (or the environment in UC) sees an honest party's output, it might use this information to distinguish the* previously simulated protocol messages *from the real messages, thereby catch the "cheating" simulator.*

In other words, the *joint distribution* of the simulated protocol messages and the simulated honest party's output might be distinguishable from the real-world view — even if they are all indistinguishable from the real world when seen individually. This is exactly why the UC-AGM-security of KOY relies on the stronger DSDH assumption: if DDH holds in the group but DSDH does not, then KOY is UC-insecure (even in the AGM) not due to (1) or (2) above, but rather due to this more "hidden" reason.

Now one might wonder what all these actually mean. It is true that in all three cases above simulation-based security fails because the simulator cannot complete certain tasks, but is this an "actual" security failure or merely breaking some UC formalism? (After all, none of these issues break game-based security.) This elicits a broader question: *Is the UC (or any simulation-based) notion even the "right" way to define security for PAKE (or perhaps for MPC in general)?* A satisfactory answer is way beyond the scope of this work, but our guess is that the "UC formalism" is there for an essential reason, namely guaranteeing security under composition: if an adversary against some protocol Π causes the simulator to fail without "actually" breaking Π, then it could probably be used to build an "actual" adversary that breaks some higher-level protocol that uses Π as a building block. This point has been made in the context of PAKE [JRX25, Section 3.4], but we believe a more general study of these "simulation-based-only attacks" is an intriguing future direction.

References

[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.

[ABK+21] Michel Abdalla, Manuel Barbosa, Jonathan Katz, Julian Loss, and Jiayu Xu. Algebraic adversaries in the universal composability framework. In *ASIACRYPT 2021*, pages 311–341, 2021.

[ABP15] Michel Abdalla, Fabrice Benhamouda, and David Pointcheval. Public-key encryption indistinguishable under plaintext-checkable attacks. In *PKC 2015*, pages 332–352, 2015.

[AFP05] Michel Abdalla, Pierre-Alain Fouque, and David Pointcheval. Password-based authenticated key exchange in the three-party setting. In *PKC 2005*, pages 65–84, 2005.

[AHH21] Michel Abdalla, Björn Haase, and Julia Hesse. Security analysis of CPace. In *ASIACRYPT 2021*, pages 711–741, 2021.

[AP05] Michel Abdalla and David Pointcheval. Simple password-based encrypted key exchange protocols. In *CT-RSA 2005*, pages 191–208, 2005.

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

[Ber13] Daniel J. Berstein. The fundamental goal of "provable security". https://cr.yp.to/talks/2013.01.23/slides.pdf, 2013.

[BFL20] Balthazar Bauer, Georg Fuchsbauer, and Julian Loss. A classification of computational assumptions in the algebraic group model. In *CRYPTO 2020*, pages 121–151, 2020.

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

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

[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.

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

[CS98] Ronald Cramer and Victor Shoup. A practical public key cryptosystem provably secure against adaptive chosen ciphertext attack. In *CRYPTO 1998*, pages 13–25, 1998.

[CS03] Ronald Cramer and Victor Shoup. Design and analysis of practical public-key encryption schemes secure against adaptive chosen ciphertext attack. *SIAM Journal on Computing*, 33(1):167–226, 2003.

[DL24] Dennis Dayanikli and Anja Lehmann. Provable security analysis of the Secure Remote Password protocol. In *CSF 2024*, pages 620–635, 2024.

[FKL18] Georg Fuchsbauer, Eike Kiltz, and Julian Loss. The algebraic group model and its applications. In *CRYPTO 2018*, pages 33–62, 2018.

[Gen08] Rosario Gennaro. Faster and shorter password-authenticated key exchange. In *TCC 2008*, pages 589–606, 2008.

[GJO10] Vipul Goyal, Abhishek Jain, and Rafail Ostrovsky. Password-authenticated session-key generation on the internet in the plain model. In *CRYPTO 2010*, pages 277–294, 2010.

[GK10] Adam Groce and Jonathan Katz. A new framework for efficient password-based authenticated key exchange. In *CCS 2010*, pages 516–525, 2010.

[GL01] Oded Goldreich and Yehuda Lindell. Session-key generation using human passwords only. In *CRYPTO 2001*, pages 408–432, 2001.

[GL03] Rosario Gennaro and Yehuda Lindell. A framework for password-based authenticated key exchange. In *EUROCRYPT 2003*, pages 524–543, 2003.

[HL19] Björn Haase and Benoît Labrique. AuCPace: Efficient verifier-based PAKE protocol tailored for the IIoT. In *CHES 2019*, pages 1–48, 2019.

[Jab97] David P. Jablon. Extended password key exchange protocols immune to dictionary attacks. In *WETICE 1997*, pages 248–255, 1997.

[Jar22] Stanislaw Jarecki. Password authenticated key exchange: Protocols and security models. In David Pointcheval, editor, *Asymmetric Cryptography*, pages 213–255. Wiley, 2022.

[JG04] Shaoquan Jiang and Guang Gong. Password based key exchange with mutual authentication. In *SAC 2004*, pages 267–279, 2004.

[JKX18] Stanislaw Jarecki, Hugo Krawczyk, and Jiayu Xu. OPAQUE: An asymmetric PAKE protocol secure against pre-computation attacks. In *EUROCRYPT 2018*, pages 456–486, 2018.

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

[KMTG05] Jonathan Katz, Philip D. MacKenzie, Gelareh Taban, and Virgil D. Gligor. Two-server password-only authenticated key exchange. In *ACNS 2005*, pages 1–16, 2005.

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

[KV09] Jonathan Katz and Vinod Vaikuntanathan. Smooth projective hashing and password-based authenticated key exchange from lattices. In *ASIACRYPT 2009*, pages 636–652, 2009.

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

[NV04] Minh-Huyen Nguyen and Salil P. Vadhan. Simpler session-key generation from short random passwords. In *TCC 2004*, pages 428–445, 2004.

[PS04] Manoj Prabhakaran and Amit Sahai. New notions of security: Achieving universal composability without trusted setup. In *STOC 2004*, pages 242–251, 2004.

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

[Sho01] Victor Shoup. A proposal for an ISO standard for public key encryption, 2001. https://eprint.iacr.org/2001/112.

[XZ25] Jiayu Xu and Zhiyuan Zhao. Just how secure is SRP, really?, 2025. https://eprint.iacr.org/2025/1869.

A Security of the Cramer–Shoup Encryption Scheme with Delayed Reveal of the Secret Key

In this section we prove the delayed-reveal-CCA-security of the Cramer–Shoup encryption scheme. We first start by formally defining the delayed-reveal-CCA game.

A.1 Security Against Delayed-Reveal Chosen Ciphertext Attack

Stage 1: The adversary queries a *key generation oracle*. The key generation oracle computes $(\mathsf{PK}, \mathsf{SK}) \leftarrow \mathsf{PKE.KeyGen}(1^\lambda)$ and responds with PK.

Stage 2: The adversary makes a sequence of calls to a decryption oracle. For each decryption oracle query, the adversary submits a ciphertext ψ, and the decryption oracle responds with $\mathsf{PKE.Dec}(\mathsf{SK}, \psi)$.

Stage 3: The adversary submits two messages $m_0, m_1 \in \mathsf{PKE.MsgSpace}_{\lambda,\mathsf{PK}}$ to an encryption oracle. We require that $|m_0| = |m_1|$.
On input (m_0, m_1), the encryption oracle computes

$$b \xleftarrow{\$} \{0,1\}; \psi^* \leftarrow \mathsf{PKE.Enc}(\mathsf{PK}, m_b)$$

and responds with the target ciphertext ψ^*.

Stage 4: The adversary continues to make calls to the decryption oracle, subject only to the restriction that a submitted ciphertext ψ is not equal to ψ^*.

Stage 5: The adversary submits a halt statement to an *auxilliary oracle*, which responds with some auxiliary information aux. From this point on the adversary no longer has access to the decryption oracle, and is not able to make any further queries.

Stage 6: The adversary outputs $\hat{b} \in \{0,1\}$.

Figure 7: The Delayed-Reveal-CCA Game with Auxiliary Information aux.

We define the delayed-reveal-CCA advantage with auxiliary information aux of the adversary $\mathcal{A}$ against PKE, denoted as $\mathsf{AdvDRCCA}_{\mathcal{A}}(\lambda)$, to be $2|\Pr[\hat{b} = b] - 1/2|$ in the attack game in Fig. 7 with corresponding value of aux.

Definition A.1 (Security Against Delayed-Reveal-Chosen Ciphertext Attack (DR-CCA))**.** *Let* PKE *be any public-key encryption scheme. We say that* PKE *is delayed-reveal-CCA-secure with auxiliary information* aux *if there exists some negligible function such that for all* $\lambda \in \mathbb{Z}_{\geq 0}$*,*

$$\mathsf{AdvDRCCA}_{\mathcal{A}}(\lambda) \leq \mathsf{negl}(\lambda).$$

Note that standard CCA-security is a special case of DR-CCA-security with auxiliary information $\mathsf{aux} = \perp$.

With this in hand, we can now formally state the theorem.

Theorem A.1. *If the DSDH assumption holds for* $\mathcal{G}$ *and the hash function* H *is collision-resistant, then the Cramer–Shoup encryption scheme with labels (Fig. 1) is DR-CCA-secure with auxiliary information* $\mathsf{aux} = (x_1, x_2, y_1, y_2, g_1^{r^2}, g_2^{r^2}, (h^r \cdot m_b)^r, (cd^\alpha)^{r^2})$.

A.2 Analysis of DR-CCA Security of the Cramer–Shoup Encryption Scheme

A.2.1 Overview

Our argument proceeds very closely to the proof of the (standard) CCA-security of the Cramer–Shoup encryption scheme [CS03, Section 6.2]. The argument follows a series of games starting at game $\mathbf{G}_0$, which corresponds to the actual attack, and ending at $\mathbf{G}_8$, which is a game in which the ciphertext returned to the adversary is completely independent of the hidden bit b. In each game, b takes on identical values. We now define a few helpful variables to quantify the adversary's advantage in each game. Let T_i be the event in game $\mathbf{G}_i$ that $\hat{b} = b$. We will show for each game that $|\Pr[T_{i+1}] - \Pr[T_i]|$ is negligible. Game $\mathbf{G}_8$, which is completely independent of b, naturally has probability $\Pr[T_8] = 1/2$, because in this game the adversary can do no better than a coin toss. We also introduce the variable $\mathsf{AdvDRCCA}_i(\lambda)$ to represent the *advantage* of the adversary in game $\mathbf{G}_i$, which is defined as $2|\Pr[T_i] - 1/2|$. Ultimately our goal is to show that $\mathsf{AdvDRCCA}_0(\lambda) \leq \mathsf{negl}(\lambda)$.

The probability is taken over the following mutually independent random variables:

1. The internal coin tosses of $\mathcal{A}$.
2. The values $\mathsf{hk}, w, x_1, x_2, y_1, y_2, z_1, z_2$ generated independently by the key generation algorithm.
3. The values $b \in \{0,1\}$ and $r \in \mathbb{Z}_q$ which are generated by the encryption oracle.

Outline of hybrids. Before continuing formally, we outline a brief sketch of each of the games.

- In game $\mathbf{G}_1$ we make some technical changes to the encryption algorithm which have no effect on the adversary's view. After game $\mathbf{G}_5$, the randomness r will be largely meaningless and thus cannot be used for correct encryption. This game ensures that encryption can occur without using r.
- Game $\mathbf{G}_2$ moves part of the auxiliary information, $\mathsf{aux}_1 = (g_1^{r^2}, g_2^{r^2}, (h^r \cdot m_i)^r, (cd^\alpha)^{r^2})$ as an output of the encryption oracle. This can only increase the power of the adversary, since it is also allowed to make decryption oracle queries that involve the auxiliary information.
- Game $\mathbf{G}_3$ replaces x^{r^2} with random values, which follows from DSDH. It shows that the auxiliary information aux_1 has negligible effect on the adversary's advantage.
- Game $\mathbf{G}_4$ is a technical game that completes the argument of $\mathbf{G}_3$.
- Game $\mathbf{G}_5$ replaces u_2 with a random value, which follows from DDH. This breaks the correlation between u_1, u_2 and $h^r \cdot m_b$, allowing m_b to be replaced by a completely random value in future games.
- Game $\mathbf{G}_6$ is really the core of the proof. In this game the decryption oracle is modified so that it additionally checks whether u_2 is indeed g_2^r for some r. The indistinguishability of this game from the previous one follows from the fact that the adversary cannot with any nontrivial probability submit a valid ciphertext which does not carry the correlation (g_1, g_2, h) that is given by the public key. In particular, note that $\mathbf{G}_5$ ensures that the target ciphertext ψ^* does *not* have that particular correlation. The immediate implication is that the adversary cannot with any nontrivial probability modify the target ciphertext

ψ^* in such a way that obtains even a different, valid ciphertext, let alone one that provides information about m_b. However, the argument of indistinguishability is nontrivial and spreads across more games.

- In game $\mathbf{G}_7$, m_b is replaced by a uniformly random value. We will show that both the advantage difference in this game, and the probability that the adversary is able to come up with some "bad" ciphertext which could have potentially provided information about m_b, are negligible.

- Finally, game $\mathbf{G}_8$ bounds the probability of a "bad" ciphertext by modifying the decryption oracle, which now rejects if the adversary is able to reuse the MAC part v of the ciphertext. The indistinguishability argument shows that if the adversary could make this query, then it either (a) broke the hash function, or (b) made an extremely unlikely guess, which it simply does not have enough information to with any nontrivial probability.

A.2.2 Proof of Theorem A.1

Notation. We first describe some helpful notation, most of which is borrowed from [CS03]. Let $\mathcal{A}$ be the DR-CCA adversary. Recall that the public key in Cramer–Shoup is $(\Gamma[\mathbb{G}, g, q], \mathsf{hk}, g_1, g_2, c, d, h)$ where $g_1 = g$, and that the secret key is $(\Gamma, \mathsf{hk}, x_1, x_2, y_1, y_2, z_1, z_2)$. Let $w := \log_g g_2$ and define x, y, z as follows:

$$x := x_1 + x_2 w, y := y_1 + y_2 w, z := z_1 + z_2 w.$$

In particular, $x = \log_g c$, $y = \log_g d$ and $z = \log_g h$.

When we deal with ciphertext ψ, we also define the following values:

- $u_1, u_2, e', e, v \in \mathbb{G}$, where $\psi = (u_1, u_2, e, v)$, with $h = u_1^{z_1} u_2^{z_2}$.

- $r, \hat{r} = \log_g u_2, \alpha, r_e = \log_g e, r_v = \log_g d$, and $t = x_1 r + y_1 r\alpha + x_2 \hat{r} w + y_2 \hat{r} \alpha w$.

For the target ciphertext ψ^*, we denote each of the variables with a $*$, i.e., the corresponding values are $u_1^*, u_2^*, (e')^*, e^*, v^*, r^*, \hat{r}^*, \alpha^*, r_e^*, r_v^*, t^*$.

Hybrids. We now proceed to describe the games in detail. Game $\mathbf{G}_0$ is the original DR-CCA game.

Game $\mathbf{G}_1$. This game is the same as $\mathbf{G}_1$ of [CS03].

We modify game $\mathbf{G}_0$ to obtain the new game, which is identical except for a small modification to the encryption oracle. Instead of using the encryption algorithm as given to compute the target ciphertext ψ^*, we use a modified encryption algorithm, in which the steps **E4** and **E7** (see Fig. 1) are replaced by

E4′ : $e' := u_1^{z_1} u_2^{z_2}$.

E7′ : $v := u_1^{x_1 + y_1 \alpha} u_2^{x_2 + y_2 \alpha}$.

These changes are purely conceptual and the values of $(e')^*$ and v^* are exactly the same in both games. It follows that

$$\Pr[T_0] = \Pr[T_1].$$

Game $\mathbf{G}_2$. This game is identical to $\mathbf{G}_1$ with the following modification. We modify the encryption oracle in Stage 3 such that along with the target ciphertext ψ^*, it also outputs the

auxiliary information $\mathsf{aux}_1 = (g_1^{r^2}, g_2^{r^2}, (h^r \cdot m_b)^r, (cd^\alpha)^{r^2}) = ((u_1^*)^r, (u_2^*)^r, (e^*)^r, (v^*)^r)$. Furthermore, we also modify the auxiliary oracle such that it no longer outputs the previous string, instead outputting only $\mathsf{aux}_2 = (x_1, x_2, y_1, y_2)$.

Our analysis of this game is simple. Note that by providing $\mathcal{A}$ the values $(g_1^{r^2}, g_2^{r^2}, (h^r \cdot m_b)^r, (cd^\alpha)^{r^2})$ *before* it can no longer make decryption oracle queries, we can only increase $\mathcal{A}$'s power (and hence advantage). It follows that $\mathsf{AdvDRCCA}_1(\lambda) \leq \mathsf{AdvDRCCA}_2(\lambda)$. Rephrasing, we can see that

$$\Pr[T_1] \leq \Pr[T_2].$$

Game $\mathbf{G}_3$. In this game we again modify the encryption oracle in Stage 3. Instead of outputting the target ciphertext $(\psi^*, \mathsf{aux}_1) = ((u_1^*, u_2^*, e^*, v^*), (g_1^{r^2}, g_2^{r^2}, (h^r \cdot m_b)^r, (cd^\alpha)^{r^2}))$, the oracle samples $s \xleftarrow{\$} \mathbb{Z}_q$ and outputs $(\psi^*, \mathsf{aux}_1) = ((u_1^*, u_2^*, e^*, v^*), (g_1^s, g_2^{r^2}, (h^r \cdot m_b)^r, (cd^\alpha)^{r^2}))$.

Recall that $g_1 = g$ and $u_1^* = g_1^r$, so the only difference between the games $\mathbf{G}_2$ and $\mathbf{G}_3$ is that the tuple $(g, u_1^*, \mathsf{aux}_1[0])$ is a uniformly distributed SDH tuple in $\mathbf{G}_2$ while it is a uniformly distributed tuple from $\mathbb{Z}_q^2$ in $\mathbf{G}_3$. It is thus immediately clear that the indistinguishability of the two games follows from the DSDH assumption. More specifically, we show the following.

Lemma A.1. *There exists some PPT algorithm $\mathcal{A}_{\mathsf{DSDH}}$ such that*

$$|\Pr[T_3] - \Pr[T_2]| \leq \mathsf{AdvDSDH}_{\mathcal{A}_{\mathsf{DSDH}},\mathcal{G}}(\lambda|\Gamma).$$

Proof. We will describe $\mathcal{A}_{\mathsf{DSDH}}$ in detail. The algorithm takes in as input some tuple (g^a, g^b), and interacts with the DR-CCA adversary $\mathcal{A}$. First, it begins by computing

$$\mathsf{hk} \xleftarrow{\$} \mathsf{HF.Keyspace}_{\lambda,\Gamma}; w \xleftarrow{\$} \mathbb{Z}_q^*; x_1, x_2, y_1, y_2, z_1, z_2 \xleftarrow{\$} \mathbb{Z}_q; c := g^{x_1 + wx_2}; d := g^{y_1 + wy_2}; h := g^{z_1 + wz_2}.$$

Using the above, it generates a public key $\mathsf{PK} = (\Gamma, \mathsf{hk}, g, g^w, c, d, h)$ and a secret key $\mathsf{SK} = (\Gamma, \mathsf{hk}, x_1, x_2, y_1, y_2, z_1, z_2)$, and passes PK to $\mathcal{A}$. We now show the behaviour of $\mathcal{A}_{\mathsf{DSDH}}$ on encryption and decryption queries. Whenever it receives a ciphertext of the form $\psi = (u_1, u_2, e, v)$ to the decryption oracle, it uses SK to decrypt and outputs either $\perp$ or some m. When $\mathcal{A}$ submits (m_0, m_1) to the encryption oracle, $\mathcal{A}_{\mathsf{DSDH}}$ samples $b \xleftarrow{\$} \{0,1\}$ and computes

$$u_1^* := g^a; u_2^* := (g^a)^w; e^* := (u_1^*)^{z_1}(u_2^*)^{z_2} \cdot m_b; v^* := \mathsf{HF}_{\mathsf{hk}}^{\lambda,\Gamma}(u_1^*, u_2^*, e^*); v^* := (u_1^*)^{x_1 + v^* y_1}(u_2^*)^{x_2 + v^* y_2}$$

and outputs the ciphertext-auxiliary information pair $(\psi^*, \mathsf{aux}_1) := ((u_1^*, u_2^*, e^*, v^*), (g^b, (g^b)^w, (e^*)^2), (v^*)^2)$. On input halt to the auxiliary oracle, $\mathcal{A}_{\mathsf{DSDH}}$ outputs (x_1, x_2, y_1, y_2).

When $\mathcal{A}$ outputs bit $\hat{b}$, $\mathcal{A}_{\mathsf{DSDH}}$ outputs 1 if $\hat{b} = b$ and 0 otherwise. It is clear from the above simulation that for any fixed λ and Γ,

$$\Pr[T_2] = \Pr[\mathcal{A}_{\mathsf{DSDH}}(g^a, g^b) = 1 : a \xleftarrow{\$} \mathbb{Z}_q, b := a^2]$$
$$\Pr[T_3] = \Pr[\mathcal{A}_{\mathsf{DSDH}}(g^a, g^b) = 1 : a, b \xleftarrow{\$} \mathbb{Z}_q].$$

It immediately follows that

$$|\Pr[T_3] - \Pr[T_2]| \leq \mathsf{AdvDSDH}_{\mathcal{A}_{\mathsf{DSDH}},\mathcal{G}}(\lambda|\Gamma). \qquad \square$$

Game $\mathbf{G}_4$. We modify the encryption algorithm in Stage 3 again. Instead of outputting the ciphertext $(\psi^*, \mathsf{aux}_1) = ((u_1^*, u_2^*, e^*, v^*), (g_1^s, g_2^{r^2}, (h^r \cdot m_b)^r, (cd^\alpha)^{r^2}))$ as before, the oracle samples $s_1, s_2 \xleftarrow{\$} \mathbb{Z}_q$ and outputs $(\psi^*, \mathsf{aux}_1) := ((u_1^*, u_2^*, e^*, v^*), (g_1^s, g_2^{s_1}, h^{s_2}, (cd^\alpha)^{s_3}))$.

The proof for $g_2^{s_1}$ and $(cd^\alpha)^{s_3}$ essentially follows the same as that of the previous game, and can be omitted.

Proving the indistinguishability of the $(h^r \cdot m_b)^r$ term in aux_1 is more complicated and requires using the power of the AGM; however, it similarly reduces to the DSDH assumption. We provide a sketch of the reduction as follows. The DSDH adversary proceeds exactly as in Lemma A.1 (with the required changes to aux_1, ie. $g_1^{r^2}, g_2^{r^2}$ and $(cd^\alpha)^{r^2}$ have already been replaced with random group elements as a result of the previous hybrids). The difference is that the algebraic adversary $\mathcal{A}$ must also provide a valid group representation of m_b. Then $\mathcal{A}_{\mathsf{DSDH}}$ can simulate by adjoining $(h, h^r \cdot m_b, k \cdot m_b^r)$ and proceed exactly as in the previous game, winning with the same advantage; the last term can be calculated because the algebraic representation allows the adversary to determine m_b as a factor of h^r.

Game $\mathbf{G}_5$. This is game $\mathbf{G}_2$ of the proof of [CS03]. We modify the encryption oracle in Stage 3, replacing step **E3** of the encryption algorithm with

$$\mathbf{E3'}: \hat{r} \xleftarrow{\$} \mathbb{Z}_q \setminus \{r\}; u_2 := g_2^{\hat{r}}.$$

In the games up to $\mathbf{G}_4$ we had $r^* = \hat{r}^*$, however in game $\mathbf{G}_5$ r^* and $\hat{r}^*$ are independent of each other except that they cannot be equal. Note, however, that like in the previous game, we have a tuple (g_2, u_1^*, u_2^*) which in game $\mathbf{G}_4$ is uniform DH tuple while in game $\mathbf{G}_5$ is uniform tuple in $\mathbb{G}^3$. This leads to the following lemma, which says that any distinguisher for the two games immediately gives a distinguisher for the DDH instance.

Lemma A.2. *There exists some PPT algorithm $\mathcal{A}_{\mathsf{DDH}}$ such that*

$$|\Pr[T_5] - \Pr[T_4]| \leq \mathsf{AdvDDH}_{\mathcal{A}_{\mathsf{DDH}},\mathcal{G}}(\lambda|\Gamma).$$

Proof. The proof of this lemma is borrowed from [CS03] and follows the same broad argument as that of Lemma A.1. The DDH distinguisher $\mathcal{A}_{\mathsf{DDH}}$ is initiated with a tuple $(g_2, u_1, u_2) \in \mathbb{G}^3$.

The DR-CCA adversary $\mathcal{A}$ is used in the same way as Lemma A.1, with the decryption and auxiliary oracles having the same description. The only difference is how the encryption oracle computes the target ciphertext. In this case, when $\mathcal{A}_{\mathsf{DDH}}$ receives (m_0, m_1), it samples $b \xleftarrow{\$} \{0,1\}$ and computes

$$e^* := (u_1^*)^{z_1}(u_2^*)^{z_2} \cdot m_b; v^* := \mathsf{HF}_{\mathsf{hk}}^{\lambda,\Gamma}(u_1^*, u_2^*, e^*); v^* := (u_1^*)^{x_1+v^*y_1}(u_2^*)^{x_2+v^*y_2}.$$

The auxiliary information aux_1 is sampled in the same manner, with g^s and $g^{s'}$ uniformly random elements of $\mathbb{G}$.

When $\mathcal{A}$ outputs bit $\hat{b}$, $\mathcal{A}_{\mathsf{DDH}}$ outputs 1 if $\hat{b} = b$ and 0 otherwise. It is clear from the above simulation that for any fixed λ and Γ,

$$\Pr[T_4] = \Pr[\mathcal{A}_{\mathsf{DDH}}(g^a, g^b, g^c) = 1 : a, b \xleftarrow{\$} \mathbb{Z}_q, c := ab],$$
$$\Pr[T_5] = \Pr[\mathcal{A}_{\mathsf{DDH}}(g^a, g^b, g^c) = 1 : a, b \xleftarrow{\$} \mathbb{Z}_q, c \xleftarrow{\$} \mathbb{Z}_q \setminus \{ab\}].$$

It immediately follows that

$$|\Pr[T_5] - \Pr[T_4]| \leq \mathsf{AdvDDH}_{\mathcal{A}_{\mathsf{DDH}},\mathcal{G}}(\lambda|\Gamma). \qquad \square$$

Game $\mathbf{G}_6$. This is game $\mathbf{G}_3$ of the proof of [CS03]. In this game, we modify the decryption oracle by replacing steps **D4** and **D5** with:

D4′: Test if $u_2 = u_1^w$ and $v = u_1^{x+y\alpha}$. Output reject and halt if this is not the case.

D5′: $h := u_1^z$.

We first notice that the decryption oracle does not make any direct use of x_i, y_i, z_i. It is easy to see that the decryption oracle of game $\mathbf{G}_5$ correctly answers all queries that $\mathbf{G}_6$ does. However, it is possible that $\mathbf{G}_6$ may reject certain queries. Let R_6 be the event that there is some query which is asked to $\mathbf{G}_6$ which would have been answered correctly by $\mathbf{G}_5$, but is rejected by $\mathbf{G}_6$. Note that $\mathbf{G}_5$ and $\mathbf{G}_6$ are identical unless R_6 occurs, so

$$|\Pr[T_6] - \Pr[T_5]| \leq \Pr[R_6]$$

and it suffices to bound $\Pr[R_6]$. This will be done in games $\mathbf{G}_7$ and $\mathbf{G}_8$.

Game $\mathbf{G}_7$. This game is identical to $\mathbf{G}_6$, except that in this game the message encrypted in the target ciphertext is randomly selected instead of being m_b. In particular, we modify the encryption oracle and replace **E5** with

E5′: $r' \xleftarrow{\$} \mathbb{Z}_q$; $e := g^{r'}$.

It follows that $\Pr[T_7] = 1/2$, because now b is not used anywhere in the game, so $\mathcal{A}$'s output bit $\hat{b}$ is independent of it. We define event R_7 in $\mathbf{G}_7$ which is the same as R_6 in $\mathbf{G}_6$, i.e., some ciphertext ψ is submitted to the decryption oracle which is rejected by **D4′** but would have passed **D4**.

Lemma A.3. *We have*

$$\Pr[T_6] = \Pr[T_7],$$
$$\Pr[R_6] = \Pr[R_7].$$

Before we show the proof, we recall [CS03, Lemma 9].

Lemma A.4. *Let k, n be integers with $1 \leq k \leq n$ and let K be a finite field. Consider a probability space with random variables $\vec{\alpha} \in K^{n\times 1}, \vec{\beta} = (\beta_1, \ldots, \beta_k)^\top \in K^{k\times 1}, \vec{\gamma} \in K^{k\times 1}$, and $M \in K^{k\times n}$, such that $\vec{\alpha}$ is uniformly distributed over $K^{n\times 1}$, $\vec{\beta} = M\vec{\alpha} + \vec{\gamma}$, and for $1 \leq i \leq k$, the ith rows of M and γ are determined by $\beta_1, \ldots, \beta_{i-1}$.*

Then conditioning on any fixed values of $\beta_1, \ldots, \beta_{k-1}$ such that the resulting matrix M has rank k, the value of β_k is uniformly distributed over K in the resulting probability space.

(We will actually require a less general version of this lemma in the proof below.)

Proof (of Lemma A.3). Our proof is essentially the same as that of [CS03], however the presence of the auxiliary information requires a more subtle analysis.

Consider the variable $X = (\mathsf{coins}, \mathsf{hk}, w, x_1, x_2, y_1, y_2, b, r^*, \hat{r}^*)$, where coins is the internal randomness of $\mathcal{A}$, and the quantity z (recall that z is defined as $z_1 + z_2 w$). Note that X has the same values in $\mathbf{G}_6$ and $\mathbf{G}_7$.

Consider also the quantity r^*, which takes different values in $\mathbf{G}_6$ and $\mathbf{G}_7$. We call these values r_6^* and r_7^* respectively. It is clear that both R_6 and T_6 are functions of X, z and r_6^*. Also, the events R_7 and T_7 have the same functional dependence on X, z and r_7^*. Thus, showing that the distributions

$$(X, z, r_6^*) \cong (X, z, r_7^*)$$

is enough to show the lemma. Now observe that if X and z are fixed, r_7^* is uniform over $\mathbb{Z}_q$. Furthermore, note that the uniformity of r_7^* is *completely* independent of $\mathsf{aux} = (x_1, x_2, y_1, y_2)$, since e is completely independent of aux as well. Hence, knowledge of aux has no bearing on the independence of r_7^*, and it remains so even if aux is public.

We now show that the same is true for r_6^*. Consider

$$\begin{pmatrix} z \\ r_6^* \end{pmatrix} = \begin{pmatrix} 1 & w \\ r^* & w\hat{r}^* \end{pmatrix} \cdot \begin{pmatrix} z_1 \\ z_2 \end{pmatrix} + \begin{pmatrix} 0 \\ \log_g m_b \end{pmatrix}.$$

Let $M = \begin{pmatrix} 1 & w \\ r^* & w\hat{r}^* \end{pmatrix}$. Then M is fixed conditioned on X, but z_1 and z_2 are independently distributed over $\mathbb{Z}_q$ (even in the knowledge of aux). Furthermore, $\det(M) = w(\hat{r}^* - r^*) \neq 0$ (recall that the change in $\mathbf{G}_5$ guarantees that $\hat{r}^* \neq r^*$). The lemma then follows from Lemma A.4. □

Game $\mathbf{G}_8$. This game is the same as $\mathbf{G}_7$, with a small modification to the decryption oracle. We modify it so that it applies a special rejection rule: if $\mathcal{A}$ submits a ciphertext ψ for decryption at a point *after* the encryption oracle has been invoked, such that $(u_1, u_2, e) \neq (u_1^*, u_2^*, e^*)$ but $v = v^*$, then the decryption oracle outputs reject and halts even before executing $\mathbf{D4}'$.

We define two events:

1. C_8, which is the event that $\mathcal{A}$ submits a ciphertext which is rejected according to the special rule.
2. R_8, which is that some ciphertext ψ is submitted which passes the special rule, however it is rejected by $\mathbf{D4}'$, and would have been accepted by $\mathbf{D4}$.

Note that games $\mathbf{G}_7$ and $\mathbf{G}_8$ are identical unless C_8 occurs, so

$$|\Pr[R_8] - \Pr[R_7]| \leq \Pr[C_8].$$

We will show two lemmas that complete the proof. The first lemma shows that the special rejection rule can be broken if the underlying hash is broken, while the second shows that breaking the rule without breaking the hash requires an information-theoretically unlikely guess.

Lemma A.5. *There is some probabilistic polynomial-time algorithm $\mathcal{A}_{\mathsf{HASH}}$ such that*

$$\Pr[C_8] \leq \mathsf{AdvCRH}_{\mathsf{HF},\mathcal{A}_{\mathsf{HASH}}}(\lambda|\Gamma).$$

Lemma A.6. *We have*

$$\Pr[R_8] \leq Q_{\mathcal{A}}(\lambda)/q,$$

where $Q_{\mathcal{A}}(\lambda)$ is number of $\mathcal{A}$'s decryption oracle queries.

The proof of Lemma A.5 is identical to the proof of [CS03, Lemma 7] and is omitted. The proof of Lemma A.6 is also similar, yet must be adapted to our setting.

Proof (of Lemma A.6). Consider the proof of [CS03, Lemma 8]. Both sub-parts of the proof rely on the fact that $\mathsf{aux}_2 = (x_1, x_2, y_1, y_2)$ is and uniformly distributed in $\mathcal{A}$'s view, and independent of the rest of the game. Clearly this is not the case if aux_2 is provided as auxiliary information. However, the proof only relies on the fact that aux_2 be uniform and independent *at the time the decryption query is made*. Indeed, by the definition of the DR-CCA game, it is clear that once the auxiliary oracle is queried, no more decryption queries are allowed to be made. Hence, at the time any decryption query is made, aux_2 is uniform and independent, and the proof of [CS03, Lemma 8] follows without any additional concerns. □

In sum, we have shown that $|\Pr[R_8] - \Pr[R_6]|$ and $\Pr[R_8]$ are both negligible in λ, so $\Pr[R_6]$ is negligible in λ. This in turn implies that $|\Pr[T_0] - \Pr[T_7]|$ is negligible in λ; but $\Pr[T_7] = 1/2$, so $\mathsf{AdvDRCCA}_0(\lambda) = 2|\Pr[T_0] - 1/2|$ is negligible in λ. This completes the proof.