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

Naman Kumar*
Oregon State University

Jiayu Xu†
Oregon State University

February 1, 2025

Abstract

We show that the PAKE protocol of Katz, Ostrovsky and Yung (2001) is not UC-secure in the plain model. We provide the first proof of UC-security of KOY in the AGM under the hardness of the Square Diffie-Hellman (SDH) assumption.

*Email: kumarnam@oregonstate.edu
†Email: xujiay@oregonstate.edu

Contents

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 used. 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 "enhances" the entropy of a 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)*.

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 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. (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, 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. *EKE-type protocols*: UC-security requires a polynomial-time 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), 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.
   The other possibility is to analyze the 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* $\lambda_1, \ldots, \lambda_n$ such that

   $$Y = X_1^{\lambda_1} \cdots X_n^{\lambda_n},$$

   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.[1] 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,

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

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.[2] 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 complete, 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:

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.[3] 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.[4] 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, *under a stronger assumption*. 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 simulating 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 does not need to proceed any further if its password guess is correct; this possibility is accounted for in the adversary's advantage allowed in the game-based definition.)

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, there is 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.

[2] KOY requires 3 group exponentiations + 3 multi-exponentiations per party; the protocol in [CHK+05] requires around 30 group exponentiations per party.

[3] 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.

[4] 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].

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 guarantee; 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 (in particular, 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 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.[5]

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 exact security 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 theoretical 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 jargons 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 Figure 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 whose corresponding secret key is unknown.

Call the two protocol parties client and server. 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. Note that A, B, $C' = C/\mathsf{pw}$, D are all of the form g^{r_1} where g is some group element that server can compute, so they also serve as a message in a Diffie–Hellman-like protocol.

[5] 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.

- Upon receiving $(\mathsf{VK}, A, B, C, D)$, server samples its "Diffie–Hellman exponents" (x_2, y_2, z_2, w_2), and computes $E = g_1^{x_2} g_2^{y_2} h^{z_2} (cd^{\alpha})^{w_2}$.[6] Furthermore, server encrypts pw with label (msg_1, E) using Cramer–Shoup encryption as done by client 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 "Diffie–Hellman 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.

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

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

and

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

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 $X_1 = E^{r_1} = A^{x_2} B^{y_2} (C/\mathsf{pw})^{z_2} D^{w_2}$ has the following property: given E (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; otherwise X_1 is a uniformly random group element.[7] 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 pass msg_2 and msg_3 without modification as long as it passes msg_1. This is to prevent a man-in-the-middle adversary from gaining information by passing all ciphertexts but modifying the rest of the messages. Specifically, (if the signature is removed) consider an adversary that passes $\mathsf{msg}_1 = (A, B, C, D)$, 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 $\mathsf{sk}_S = \mathsf{sk}_C^2$. 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 plugging in its own verification key (and thus knowing the corresponding secret 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.

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

[7] Using the terminology of SPHF: (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 $\mathsf{Hash}_{(x_2,y_2,z_2,w_2)}(m) = A^{x_2} B^{y_2} (C/m)^{z_2} D^{w_2}$ and the projection function is defined as $\mathsf{HashProj}_E(r_1) = E^{r_1}$.

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.

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 $\mathsf{msg}_1 = (\mathsf{VK}, A, B, C, D)$. 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 $h = g_1^\ell$ — in other words, $\mathcal{S}$ chooses the "CRS trapdoor" $\ell = \log_{g_1} h$. After receiving the NewSession command from $\mathcal{F}_{\mathsf{pake}}$, $\mathcal{S}$ must simulate client's first message msg_1. Since $\mathcal{S}$ does not know the password, at this point it must (effectively) guess some pw' at random; that is, in msg_1, $C = h^{r_1} \cdot \mathsf{pw}'$ where 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^ℓ. Once this has been done, $\mathcal{S}$ can send a TestPwd command to the UC PAKE functionality $\mathcal{F}_{\mathsf{pake}}$ 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).[8] 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

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

and

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

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* 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 $A^{x_2} B^{y_2} (C/\mathsf{pw}')^{z_2} D^{w_2}$, 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 $(\mathsf{pw}/\mathsf{pw}')^{z_2}$ in order to compute the correct sk, which is infeasible unless $\mathsf{pw}' = \mathsf{pw}$ (whose probability is at most $1/|\mathcal{D}|$).

[8] A TestPwd *must* be run, since we require that msg_1 and msg_2 together with the randomness of the client and $\mathcal{A}$ together determine sk; allowing the simulation to proceed without a TestPwd would result in $\mathcal{F}_{\mathsf{pake}}$ outputting a uniformly random key.

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 x_2, y_2, z_2, w_2 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; A, B, C, D from msg_1; and pw. An algebraic $\mathcal{Z}$ must "explain" how E is computed; for now let's ignore A, B, C, D, pw 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 chose the wrong pw' with high probability while generating A, B, C, D; however, after extracting the correct pw from msg_2, $\mathcal{S}$ can still compute X_1 as $A^{x_2} B^{y_2} (C/\mathsf{pw})^{z_2} D^{w_2}$, since now it sees the algebraic coefficients x_2, y_2, z_2, w_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

$$E = g_1^{x_2} g_2^{y_2} h^{z_2} c^{w_2} d^{v_2} A^{x'_2} B^{y'_2} C^{z'_2} D^{w'_2} \mathsf{pw}^{p_2}.$$

In the real world client would compute

$$\begin{aligned} X_1 &= E^{r_1} \\ &= g_1^{r_1 x_2} g_2^{r_1 y_2} h^{r_1 z_2} c^{r_1 w_2} d^{r_1 v_2} A^{r_1 x'_2} B^{r_1 y'_2} C^{r_1 z'_2} D^{r_1 w'_2} \mathsf{pw}^{r_1 p_2} \\ &= g_1^{r_1 x_2} g_2^{r_1 y_2} \left(\frac{C}{\mathsf{pw}}\right)^{z_2} c^{r_1 w_2} d^{r_1 v_2} A^{r_1 x'_2} B^{r_1 y'_2} C^{r_1 z'_2} D^{r_1 w'_2} \mathsf{pw}^{r_1 p_2}. \end{aligned}$$

Again, in the ideal world $\mathcal{S}$ can compute X_1 according to the last equation; the key point is that given (A, B, C, D), the only place where the "wrong" password pw' is used lies in h, so we only need to "correct" h^{r_1} from C/pw' to C/pw. A difference is that now the expression of X_1 involves the Cramer–Shoup randomness r_1, which is not a problem for $\mathcal{S}$ as $\mathcal{S}$ chose r_1 itself while sending msg_1. (Of course, we have $A = g_1^{r_1}$, so the $g_1^{r_1 x_2} A^{r_1 x'_2}$ part can be rewritten as $A^{x_2 + r_1 x'_2}$, and so on. But this simplification is not necessary.)

Further subtleties in UC. As we have just seen, one critical difference between game-based security and UC-security is that in UC indistinguishability between the real view and the simulated view must remain *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 more subtle: 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 $\mathsf{sk}_C = X_1 X_2$ 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 sk_C / 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[9], 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" $\mathsf{pw}' \neq \mathsf{pw}$ to encrypt while simulating msg_1. In fact, it seems that for the KOY protocol to be UC-AGM-secure, we need 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.[10]

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 sk_C. (This step is not needed in the game-based proof).[11]

   c^{r_1} is needed only in step (c), which happens after all decryption oracle queries have been 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 pw' as C/X_1 and check if $\mathsf{pw}' = \mathsf{pw}$; in the real world this equation holds, whereas in the ideal world the simulator $\mathcal{S}$ would be "caught" using a wrong 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/pw — which is equal to $h^{r_1} \cdot (\mathsf{pw}'/\mathsf{pw})$ — instead of h^{r_1}, and if $\mathcal{Z}$ computes C/X_1, the result would be exactly pw (rather than pw'). In general, the h term in the algebraic expression of E is a special case where $\mathcal{S}$ needs to use $h^{r_1} \cdot (\mathsf{pw}'/\mathsf{pw})$ instead of h^{r_1}, i.e., "pretends" that h^{r_1} is C/pw rather than C/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 sending a message that contains an incorrect password guess, or modifying the signature) is essentially identical to the game-based proof.

[9] The 2-DL problem is: given (g^x, g^{x^2}) for $x \leftarrow \mathbb{Z}_q$, 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].

[10] DSDH implies 2-DL as given (g^x, Y) where Y is either g^{x^2} or a 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.

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

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 $\mathbb{G}$. 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 $\bar{g} \xleftarrow{\$} \bar{\mathbb{G}}$, 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 $\lambda \in \mathbb{Z}_{>0}$, $\mathsf{AdvDDH}_{\mathcal{G},\mathcal{A}}(\lambda|\Gamma) \le \mathsf{negl}(\lambda)$ (resp. $\mathsf{AdvDSDH}_{\mathcal{G},\mathcal{A}}(\lambda|\Gamma) \le \mathsf{negl}(\lambda)$).

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. 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.
- **One-time security:** There exists a negligible function negl such that for every PPT algorithm $\mathcal{A}$,

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

  where $\mathcal{A}$ makes only a single query to $\mathsf{Sign}_{\mathsf{SK}}(\cdot)$ and σ was not an output of $\mathsf{Sign}_{\mathsf{SK}}(m)$.

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 $\lambda \in \mathbb{Z}_{\geq 0}$, $\Gamma \in [S_\lambda]$ and $\mathsf{hk} \in [\mathsf{HF.Keyspace}_{\lambda,\Gamma}]$ where each $\mathsf{HF}^{\lambda,\Gamma}_{\mathsf{hk}}$ maps a tuple ρ of group elements to an element of $\mathbb{Z}_q$. Furthermore, the algorithm that outputs $\mathsf{HF}^{\lambda,\Gamma}_{\mathsf{hk}}(\rho)$ 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

$$\mathsf{AdvCRH}_{\mathsf{HF},\mathcal{A}}(\lambda|\Gamma) := \Pr[x \neq x' \wedge \mathsf{HF}^{\lambda,\Gamma}_{\mathsf{hk}}(x) = \mathsf{HF}^{\lambda,\Gamma}_{\mathsf{hk}}(x') : \mathsf{hk} \xleftarrow{\$} \mathsf{HF.Keyspace}_{\lambda,\Gamma}, (x, x') \leftarrow \mathcal{A}(1^\lambda, \Gamma, \mathsf{hk})]$$

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

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

where $\Gamma \xleftarrow{\$} [S_\lambda]$.

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

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

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

$$\mathsf{PK} = (\Gamma, \mathsf{hk}, g_1, g_2, c, d, h) \in [\mathcal{G}_\lambda] \times [\mathsf{HF.Keyspace}_{\lambda,\Gamma}] \times \mathbb{G}^4,$$

a message $m \in G$ 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 reject and halt if ψ is not of this form.
D2: Test if $u_1, u_2, e \in \mathbb{G}$; output reject 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 reject 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 $(\mathsf{NewSession}, sid, \mathsf{P}, \mathsf{P}', \mathsf{pw}, \mathsf{role})$ from P, send $(\mathsf{NewSession}, sid, \mathsf{P}, \mathsf{P}', \mathsf{role})$ to Sim. Furthermore, if this is the first $\mathsf{NewSession}$ message for sid, or this is the second $\mathsf{NewSession}$ 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 $(\mathsf{TestPwd}, sid, \mathsf{P}, \mathsf{pw}^*)$ from Sim, 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 $\mathsf{compromised}$ and send "correct guess" to Sim.
  - If $\mathsf{pw}^* \neq \mathsf{pw}$, then mark the record $\mathsf{interrupted}$ and send "wrong guess" to Sim.
- On $(\mathsf{NewKey}, sid, \mathsf{P}, \mathsf{sk}^* \in \{0,1\}^\lambda)$ from Sim, 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 (sid, sk) to P, where sk is defined as follows:
  - If the record is $\mathsf{compromised}$, or either P or P' is corrupted, then set $\mathsf{sk} := \mathsf{sk}^*$.
  - If the record is fresh, a key (sid, 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 $\mathsf{sk} \xleftarrow{\$} \{0,1\}^\lambda$.

  Finally, mark the record $\mathsf{completed}$.

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 is 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. 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 briefly describe 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 $\mathsf{TestPwd}$ 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.
- In the case of an eavesdropping attack with *incorrect password*, it is true that $\mathsf{pw} \neq \mathsf{pw}'$ and no $\mathsf{TestPwd}$ is sent. In this case, the two parties' keys sk and sk' are independently random.

- In 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 some target message such as ∇, 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.

- If the session is interrupted, this corresponds to a failed online guessing attack by the adversary. In this case, the session keys of P and P' are uniform and independent, and the adversary gains no knowledge of either.

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

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

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.

3.2 KOY is UC-Insecure

Theorem 3.1. *Assuming the hardness of fixed-CDH, the protocol of [KOY] does not UC-realize $\mathcal{F}_{\mathsf{pake}}$ in the $\mathcal{F}_{\mathsf{crs}}$-hybrid model.*

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. At a high level, we will show that any simulator that successfully simulates the protocol against $\mathcal{Z}$ in the ideal world can be used to solve arbitrary instances of fixed-CDH.

Assume that there exists a negligible function $\varepsilon := \varepsilon(\lambda)$ such that there exists a "successful" simulator $\mathcal{S}$ for which $\mathcal{Z}$ outputs 1 with probability $1 - \varepsilon$ in the ideal world. First, assume that CDH is hard over $(\mathbb{G}, p, h)$. Consider reduction $\mathcal{R}$ that runs the simulator $\mathcal{S}$ as follows (note that $\mathcal{R}$ plays the role of the environment $\mathcal{Z}$, the PAKE functionality $\mathcal{F}_{\mathsf{pake}}$, and the dummy parties client and server combined):

0. $\mathcal{R}$ receives $\mathsf{crs} = (g_1, g_2, h, c, d)$ from $\mathcal{S}$, outputs h to its challenger, and receives (h^a, h^b) where $a, b \xleftarrow{\$} \mathbb{Z}_p$.

1. $\mathcal{R}$ sends $(\mathsf{NewSession}, \mathsf{sid}, \mathsf{client}, \mathsf{server})$ to $\mathcal{S}$.

Initialization: Let $\Gamma[\mathbb{G}, g, q] \leftarrow \mathcal{G}(1^\lambda)$. Set $g_1 := g$ and sample $g_2, c, d, h \xleftarrow{\$} \overline{\mathbb{G}}$. 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 $(\mathsf{VK}, \mathsf{SK}) \xleftarrow{\$} \mathsf{OTS.Gen}(1^\lambda)$ 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 msg_1 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 msg_2 from server, client samples $x_1, y_1, z_1, w_1 \xleftarrow{\$} \mathbb{Z}_q$ and sets

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

   and sends $\mathsf{msg}_3 := (\mathsf{sid}, K, \mathsf{sig})$ 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 msg_3 from client, server checks if $\mathsf{Vrfy}_{\mathsf{VK}}(\mathsf{msg}_1, \mathsf{msg}_2, K; \mathsf{sig}) = 1$. If not, it outputs $\mathsf{sk}_{\mathsf{server}} = \perp$. 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 CRS (g_1, g_2, h, c, d).

1. $\mathcal{Z}$ selects $\mathsf{pw} \xleftarrow{\$} \mathcal{PW}$, where $\mathcal{PW} \subseteq \mathbb{G}$ is the password dictionary. 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 $\mathcal{A}$ and samples $x_2, y_2, z_2, w_2, r_2 \xleftarrow{\$} \mathbb{Z}_q$. It then sets

$$\begin{aligned}
\alpha' &:= H(A|B|C|D) \\
E &:= g_1^{x_2} g_2^{y_2} h^{z_2} (cd^{\alpha'})^{w_2} \\
F &:= g_1^{r_2} \\
G &:= g_2^{r_2} \\
I &:= h^{r_2} \cdot \mathsf{pw} \\
\beta &:= H(\mathsf{msg}_1|E|F|G|I) \\
J &:= (cd^{\beta})^{w_2}
\end{aligned}$$

   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 $\mathcal{A}$ and $(\mathsf{sid}, \mathsf{sk})$ from client.

4. $\mathcal{Z}$ sets $C' := C/\mathsf{pw}$ and then checks if $\mathsf{Vrfy}_{\mathsf{VK}}(\mathsf{msg}_1|\mathsf{msg}_2|K, \sigma) = 1$. If yes, it computes $\mathsf{sk}_S := A^{x_2} B^{y_2} (C')^{z_2} D^{w_2} K^{r_2}$ 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 $\mathcal{Z}$.

2. $\mathcal{R}$ waits to receive $\mathsf{msg}_1 = \mathsf{sid}|\mathsf{VK}|A|B|C|D$ in response (as the first message from client to server). It then sets $\mathsf{pw} := C/h^b$ and samples $x_2, y_2, w_2, r_2 \xleftarrow{\$} \mathbb{Z}_q$, setting

$$\begin{aligned}
\alpha' &:= H(PIDs|A|B|C|D) \\
E &:= g_1^{x_2} g_2^{y_2} h^{a} (cd^{\alpha'})^{w_2} \\
F &:= g_1^{r_2} \\
G &:= g_2^{r_2} \\
I &:= h^{r_2} \cdot \mathsf{pw} \\
\beta &:= H(\mathsf{msg}_1|E|F|G|I) \\
J &:= (cd^{\beta})^{w_2}
\end{aligned}$$

   i.e., the same computation as that of the honest server with the special choice of $\mathsf{pw} = C/h^b$ and $z_2 = a$, and sends $\mathsf{msg}_2 := \mathsf{sid}|E|F|G|I|J$ to $\mathcal{S}$.

3. $\mathcal{R}$ receives $\mathsf{msg}_3 = \mathsf{sid}|K|\sigma$ (as the second message from client to server) and $(\mathsf{sid}, \mathsf{sk})$ from $\mathcal{S}$ (as client's output to $\mathcal{Z}$), and checks if $\mathsf{Vrfy}_{\mathsf{VK}}(\mathsf{msg}_1|\mathsf{msg}_2|K, \sigma) = 1$. If not, $\mathcal{R}$ aborts. Otherwise it calculates

$$h' = \frac{\mathsf{sk}}{A^{x_2} B^{y_2} D^{w_2} K^{r_2}}.$$

4. $\mathcal{R}$ outputs h'.

Note that $\mathcal{S}$'s view while interacting with $\mathcal{R}$ is identical to $\mathcal{S}$'s view in the ideal world with environment $\mathcal{Z}$ in Figure 4; the difference is that $\mathcal{Z}$ samples pw and z_2 on its own, whereas $\mathcal{R}$

sets $\mathsf{pw} = C/h^b$ and $z_2 = a$ — which cannot be detected by $\mathcal{S}$. Let $C' = C/\mathsf{pw} = h^b$ and

$$\mathsf{sk}_S = A^{x_2} B^{y_2} (C')^{z_2} D^{w_2} K^{r_2} = A^{x_2} B^{y_2} (h^b)^a D^{w_2} K^{r_2}$$

as what $\mathcal{Z}$ would compute in its step 4; $\mathcal{Z}$ outputs 1 if and only if $\mathsf{sk}_S = \mathsf{sk}$, so by our assumption on $\mathcal{S}$, in $\mathcal{R}$'s interaction with $\mathcal{S}$, $\mathsf{sk}_S = \mathsf{sk}$ with probability $1 - \varepsilon$. But this gives $h' = h^{ab}$ with probability $1-\varepsilon$, i.e., $\mathcal{R}$ wins with probability $1-\varepsilon$, contradicting the hardness of fixed-CDH. □

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 of 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 hybrids. We will show that the distinguishing advantage of any PPT environment $\mathcal{Z}$ in game $\mathbf{G}_0$ (which corresponds to the real-world attack) 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 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 NewSession message, and when client or server completes a session, the key is output to the environment.

4.2 Proof Overview

The broad strokes of our proof are as below.

0. In game $\mathbf{G}_0'$, the experiment is aborted if any of the following events happen.
   (a) A VK (which is part of msg_1) is reused for different sid's.
   (b) server generates the same msg_2 twice for different sid's.
   (c) The challenger detects at some point that the adversary has been able to produce a new signature for VK in some honestly generated msg_1 (so the adversary does not know the corresponding signing key).
   (d) At any given point, the challenger detects that a collision has occured in the hash function. In this case the challenger again sends an abort and terminates the protocol.

This is indistinguishable from the attack $\mathbf{G}_0$ because the probability of any of these events happening is unlikely due to the security of the hash function and the one-time signature scheme.

1. In game $\mathbf{G}_1$, the challenger changes the way that the CRS is generated. In this case, crs is now generated according to the Cramer–Shoup KeyGen procedure, and the challenger records the secret key, i.e., the discrete logs of h, c, d. The simulator is going to (eventually) use this to extract the adversary's password guesses.

2. In game $\mathbf{G}_2$, the challenger considers the messages msg_i^* received from the adversary, and uses the secret key recorded in $\mathbf{G}_1$ to extract a password guess. If the password guess is correct, it then computes the session key in the compromised instance and sets it to be sk_i. Computing sk_i requires us to use the algebraic exponents delivered by the adversary, i.e., this part of the proof requires the AGM.

3. In game $\mathbf{G}_3$, the challenger deals with incoming msg_3^* messages. It checks the sid of the message and in case that $\mathsf{msg}_1^* = \mathsf{msg}_1$ (i.e., the adversary did not modify the first message), it sets $\mathsf{sk}_{\mathsf{server}} := \mathsf{sk}_{\mathsf{client}}$ for the same sid, which was already previously computed (at the time that msg_3 was sent). The only situation in which $\mathbf{G}_3$ is different from $\mathbf{G}_2$ is if the adversary was able to forge some (K, sig), but this has already been ruled out in $\mathbf{G}_0'$.

4. In game $\mathbf{G}_4$, the challenger again deals with incoming msg_3^* messages. It again checks the sid of the message and in the case that $\mathsf{msg}_1^* \neq \mathsf{msg}_1$ and furthermore the message is a Cramer–Shoup ciphertext of $\mathsf{pw}_{\mathsf{server}}$, the challenger outputs a session key $\mathsf{sk}_{\mathsf{server}}$ *at random*. The proof of indistinguishability proceeds from an information-theoretic argument.

5. In game $\mathbf{G}_5$, the challenger replaces the server's Cramer–Shoup encryption of $\mathsf{pw}_{\mathsf{server}}$ (in msg_2) with an encryption of some "dummy" $g^k \notin \mathsf{Dict}$. The proof of indistinguishability follows from the DR-CCA-security of the Cramer–Shoup encryption scheme, which is discussed in Section A.

6. In game $\mathbf{G}_6$, the challenger considers msg_2^*. If $\mathsf{msg}_2^* = \mathsf{msg}_2$, then the challenger sets $\mathsf{sk}_{\mathsf{client}}$ to be random. The proof of indistinguishability is similar to that of $\mathbf{G}_3$.

7. In game $\mathbf{G}_7$, the challenger considers msg_2^* again, and if $\mathsf{msg}_2^* \neq \mathsf{msg}_2$ but the message is not an encryption of $\mathsf{pw}_{\mathsf{client}}$, it sets $\mathsf{sk}_{\mathsf{client}}$ to be random as well. The proof is again similar.

8. In game $\mathbf{G}_8$, the challenger replaces the client's Cramer–Shoup encryption of $\mathsf{pw}_{\mathsf{client}}$ with an encryption of some "dummy" $g^k \notin \mathsf{Dict}$. The proof of indistinguishability follows from the DR-CCA security of Cramer–Shoup.

9. Game $\mathbf{G}_9$ is the ideal world. We argue that the challenger in $\mathbf{G}_8$ can be interpreted as the combination of the UC PAKE functionality and a simulator.

4.3 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 attack.

Game $\mathbf{G}_0'$. In this game, we introduce an additional abort clause. If this clause is activated, then the challenger sends a message abort and halts. The abort clause is activated in the case of the following situations:

1. The verification key VK (in msg_1) is used more than once by the system. This encompasses the case when client's key generation algorithm outputs VK twice for two separate instantiations in two separate sessions.

2. server generates a msg_2 which is identical to a different msg_2 generated in a separate session.

3. Consider $(\mathsf{SK}, \mathsf{VK})$ to be honest outputs of $\mathsf{OTS.KeyGen}(1^\lambda)$ by the client. The situation is that $\mathcal{A}$ is able to output some sig' which was *not* generated by the challenger as part of msg_3, and $\mathsf{Vrfy}_{\mathsf{VK}}(\mathsf{msg}_1, \mathsf{msg}_2, K; \mathsf{sig}') = 1$.

4. A collision occurs in the hash function.

We will show that the probability of each of these cases is negligible in λ.

For (2), this is information-theoretic: the only way this can happen is if the selected exponents repeated, which occurs with negligible probability.

(4) occurs with negligible probability by the collision-resistance of the hash function HF.

Case (1) and (3) occur with negligible probability by assumption that OTS is a one-time signature scheme.

It follows that

$$\Pr[Z_0] \leq \Pr[Z_0'] + \mathsf{negl}(\lambda).$$

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 $\Gamma[\mathbb{G}, g, q] \xleftarrow{\$} [S_\lambda]$. Set $g_1 := g$ and sample $g_2 \xleftarrow{\$} \overline{\mathbb{G}}$. Sample

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

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 $\overline{\mathbb{G}}$. It follows that

$$\Pr[Z_0'] = \Pr[Z_1].$$

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

1. If $\mathsf{msg}_i^* \neq \mathsf{msg}_i$, 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*.

2. Consider $\mathsf{msg}_1^* = (\mathsf{sid}, \mathsf{VK}, A, B, C, D)$. 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.)

3. Consider $\mathsf{msg}_2^* = (\mathsf{sid}, E, F, G, I, J)$. 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 two changes regarding how the two parties' session keys are computed. After this game, the password will no longer be used in computing the session keys, except implicitly.

When the challenger receives msg_2^*, if it is adversarially generated and valid, the challenger considers $\mathcal{A}$'s algebraic exponents $(x_2, y_2, z_2, w_2, r_2)$, which were provided along with msg_2^*. Jiayu: This assumes that $\mathcal{A}$ computes E as $g_1^{x_2} g_2^{y_2} h^{z_2} c^{w_2} d^{\alpha w_2}$; what if $\mathcal{A}$ computes $E = g_1^{x_2} g_2^{y_2} h^{z_2} c^{w_2} d^{v_2} A^{x'_2} B^{y'_2} C^{z'_2} D^{w'_2} \mathsf{pw}^{p_2}$? It then computes

$$\mathsf{pw}' := \frac{I/I'}{C/C'}$$

and sets the session key $\mathsf{sk}_{\mathsf{client}} := E^{r_1} F^{x_1} G^{y_1} (I')^{z_1} J^{w_1} (\mathsf{pw}')^{z_2}$. The challenger then responds to the message with msg_3^* exactly as it does in $\mathbf{G}_1$. Jiayu: I am a bit lost here... what is C' and what is I'? Note that in the real protocol $C' = C/\mathsf{pw}_{\mathsf{server}}$ and $I' = I/\mathsf{pw}_{\mathsf{client}}$, and the two passwords might be different

When the challenger receives $\mathsf{msg}_3^* = (K, \mathsf{sig})$, it examines msg_1^* for the corresponding sid. If msg_1^* is adversarially generated and valid, it considers $\mathcal{A}$'s algebraic exponents output with msg_1^* and msg_3^* as $(x_1, y_1, z_1, w_1, r_1)$. ($r_1$ is provided together with msg_1^*, and (x_1, y_1, z_1, w_1) is provided together with msg_3^*.) If $\mathsf{Vrfy}_{\mathsf{VK}}(\mathsf{msg}_1^*, \mathsf{msg}_2, K; \mathsf{sig}) = 1$, it computes

$$C' := \frac{C}{I/h^{r_1}}$$

and sets the session key $\mathsf{sk}_{\mathsf{server}} := A^{x_2} B^{y_2} (C')^{z_2} D^{w_2} K^{r_2}$.

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 message is detected is identical. In the case such a message *is* detected, note that by correctness of the protocol of Fig. 3 we have

$$\mathsf{pw}' = \frac{I/I'}{C/C'} = 1$$
$$\frac{C}{I/h^{r_1}} = \frac{C}{\mathsf{pw}_{\mathsf{client}}}$$

and hence

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

Thus there is no difference in the distribution of sk_i between $\mathbf{G}_1$ and $\mathbf{G}_2$, and 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, it checks the corresponding msg_1^* of 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 a $\mathsf{sk}_{\mathsf{client}}$ exists. However, this follows from the fact that the protocol did not abort. By the abort clause introduced in game $\mathbf{G}_0'$, there must be some unique sid for which the VK corresponding to msg_3^* was used, and furthermore this is the same sid corresponding to msg_3^* (because a case of successful verification if the two sid's were not equal would mean a signature was forged).

Note that in $\mathbf{G}_2$ it is true that in the case of honestly generated msg_1, $\mathsf{sk}_{\mathsf{client}} = \mathsf{sk}_{\mathsf{server}}$. Hence, the following is unconditionally true:

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

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 $g = g_1$:

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

It is easy to see that these equations are linearly independent. Jiayu: You mean the independence of the two equations in the above linear system? If $A^{\chi_1+\alpha\zeta_1}B^{\chi_2+\alpha\zeta_2} \neq D$, then note that $(\chi_1+\alpha\zeta_1)+(\chi_2+\alpha\zeta_2)\log g_2 = \log(cd^{\alpha})$, while if $C/\mathsf{pw}_{\mathsf{server}} \neq A^{\kappa}$, then the independence is trivial. 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 completely independent of the value of E, and it follows that the distribution of $\mathsf{sk}_{\mathsf{server}}$ is uniform and independent of $\mathcal{A}$'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 Lemma 4.2. Jiayu: DOUBLE CHECK □

Game $\mathbf{G}_6$. In this game the challenger modifies how it determines msg_2. On receiving msg_1^*, instead of setting $I := h^r \cdot \mathsf{pw}_{\mathsf{server}}$, it sets $I := h^r \cdot g^{N+1}$ where g^{N+1} is a "dummy" plaintext that does not belong to the password dictionary (so it is guaranteed that $g^{N+1} \neq \mathsf{pw}_{\mathsf{server}}$).[12]

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

Proof. By Theorem A.1, under DSDH the Crame–Shoup with labels encryption scheme satisfies DR-CCA-security with auxiliary information $\mathsf{aux} = (x_1, x_2, y_1, y_2, g_1^{r^2}, g_2^{r^2}, (h^r \cdot m_b)^r, (cd^\alpha)^{r^2})$. We show that any environment $\mathcal{Z}$ which can distinguish between $\mathbf{G}_5$ and $\mathbf{G}_6$ can be used as a subroutine to build a reduction $\mathcal{R}$ which can break the DR-CCA-security (Fig. 7) for Cramer–Shoup with labels (Fig. 1).

$\mathcal{R}$ begins by querying the key generation oracle and receiving the public key $\mathsf{PK} = (\Gamma, \mathsf{hk}, g_1, g_2, c, d, h)$. It sets the $\mathsf{crs} = (\Gamma, g_1, g_2, h, c, d, \mathsf{hk})$ and begins a simulation of the challenger for $\mathcal{A}$ (and $\mathcal{Z}$). This simulation proceeds as in $\mathbf{G}_5$, with the following difference:

- On receiving a message msg_1^*, $\mathcal{R}$ submits $m_0 := \mathsf{pw}_{\mathsf{server}}$ (which was provided by $\mathcal{Z}$) and $m_1 := g^{N+1}$ as plaintexts 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 an adversarially generated message msg_2^*, $\mathcal{R}$ needs to determine whether this message is valid or invalid (recall that in this setting the initialization procedure has not been backdoored, so $\mathcal{R}$ cannot check whether the message is valid as what the real challenger does). To do this, it queries the decryption oracle with (F, G, I, J), and sets the message to be valid if the result is $\mathsf{pw}_{\mathsf{client}}$ and invalid otherwise. Jiayu: How is it guaranteed that $\mathcal{R}$ does not query the decryption oracle on the challenge ciphertext?
- On receiving an adversarially generated message msg_3^*, $\mathcal{R}$ again needs to determine whether msg_1^* is valid or invalid. To do so, it again queries the decryption oracle with (A, B, C, D) and sets the message to be valid if the result is $\mathsf{pw}_{\mathsf{server}}$ and invalid otherwise.
- $\mathcal{R}$ determines the algebraic exponents of K delivered by $\mathcal{A}$ using $\mathsf{msg}_3^{'*}$ and queries the auxiliary oracle to receive $\mathsf{aux} = (g_1^{r_2^2}, g_2^{r_2^2}, c^{r_2}, d^{r_2}, (h^{r_2} \cdot m_i)^{r_2}, (cd^\alpha)^{{r_2}^2})$. Using this, in the case that msg_1^* is adversarially generated and valid, it determines $\mathsf{sk}_{\mathsf{server}} = A^{x_2} B^{y_2} C^{z_2} D^{w_2} K^{r_2}$, since K is completely determined as some function of aux. It then outputs $\mathsf{sk}_{\mathsf{server}}$ to $\mathcal{Z}$. Jiayu: This might need further explanation; see comment below $\mathbf{G}_2$

$\mathcal{R}$ then outputs what $\mathcal{Z}$ outputs.

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

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

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 closely related to that of game $\mathbf{G}_4$. Note that 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,

[12] This assumes that the dictionary $\mathsf{Dict} = \{1, \ldots, N\}$ is public. If it is not, the challenger can instead sample the "dummy" plaintext $\widetilde{\mathsf{pw}}$ uniformly from $\mathbb{G}$, and $\widetilde{\mathsf{pw}} \neq \mathsf{pw}_{\mathsf{server}}$ holds with overwhelming probability.

and the proof of linear independence is also nearly identical. We can conclude that the session key is uniform 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 $\mathsf{sk}_{\mathsf{client}} \xleftarrow{\$} \mathbb{G}$.

Note that in game $\mathbf{G}_6$ I is set to $h^{r_2} \cdot g^{N+1} = g^{r_2\kappa} \cdot g^{N+1} = F^\kappa \cdot g^{N+1}$, 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 $\mathbf{G}_7$, and we get

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

Game $\mathbf{G}_9$. In this game the challenger modifies the client's response to the opening $\mathsf{NewSession}$ message. It computes $C := h^r g_1^{N+1}$, similar to how it does in game $\mathbf{G}_6$, encrypting a "dummy" password not in the dictionary.

Lemma 4.5. *Under the DSDH assumption,* $|\Pr[Z_8] - \Pr[Z_9]| \leq \mathsf{negl}(\lambda)$.

Proof. Like the proof of $\mathbf{G}_6$, we will show that any environment $\mathcal{Z}$ which can distinguish between the two games can be used to build a reduction which breaks the DR-CCA-security of Cramer–Shoup with labels.

The description of the reduction $\mathcal{R}$ is as follows. $\mathcal{R}$ begins by querying the key generation oracle and receiving the public key $\mathsf{PK} = (\Gamma, \mathsf{hk}, g_1, g_2, c, d, h)$. It sets $\mathsf{crs} := (\Gamma, g_1, g_2, h, c, d, \mathsf{hk})$ and begins a simulation of the challenger for $\mathcal{A}$ (and $\mathcal{Z}$). This simulation proceeds as in $\mathbf{G}_7$, with the following difference:

- On receiving a message $(\mathsf{NewSession}, \mathsf{sid}, \mathsf{client}, \mathsf{server}, \mathsf{pw}_{\mathsf{client}})$, $\mathcal{R}$ submits $m_0 := \mathsf{pw}_{\mathsf{client}}$ and $m_1 := g^{N+1}$ as plaintexts for the encryption oracle. It receives in return the target ciphertext ψ^*, and sets $(A, B, C, D) := \psi^*$.
- On receiving an adversarially generated message msg_2^*, the challenger needs to determine whether this message is valid or invalid (recall that in this setting the initialization procedure has not been backdoored). To do this, it queries the decryption oracle with (F, G, I, J), and sets the message to be valid if it is an encryption of $\mathsf{pw}_{\mathsf{client}}$ and invalid otherwise.
- On receiving an adversarially generated message msg_3^*, the challenger again needs to determine whether msg_1^* is valid or invalid. To do so, it again queries the decryption oracle with (A, B, C, D) and sets the message to be valid if it is an encryption of $\mathsf{pw}_{\mathsf{client}}$ and invalid otherwise.
- The simulator determines the algebraic exponents of the E delivered by the adversary using $\mathsf{msg}_2'^*$ and queries the auxiliary oracle to receive $\mathsf{aux} = (g_1^{r_1^2}, g_2^{r_1^2}, c^{r_1}, d^{r_1}, (h^{r_1} \cdot m_i)^{r_1}, (cd^\alpha)^{{r_1}^2})$. Using this, in the case that msg_2^* is adversarially generated and valid, it determines $\mathsf{sk}_{\mathsf{client}} = E^{r_1} F^{x_1} G^{y_1} (I')^{z_1} J^{w_1} (\mathsf{pw}')^{z_2}$, since E is completely determined as a function of aux. It then releases $\mathsf{sk}_{\mathsf{client}}$ to $\mathcal{Z}$.

$\mathcal{A}$ then outputs what $\mathcal{Z}$ outputs.

It is clear that the above is a correct simulation of $\mathbf{G}_7$ when m_0 is encrypted and $\mathbf{G}_8$ when m_1 is encrypted. It follows immediately that

$$|\Pr[Z_7] - \Pr[Z_8]| \leq \mathsf{AdvDRCCA}_{\mathcal{A}}(\lambda) \leq \mathsf{negl}(\lambda).$$

□

Game $\mathbf{G}_9$. In this game we introduce the simulator shown in Fig. 6. It is easy to see that the simulator does exactly what is done by the challenger of $\mathbf{G}_8$, 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 sk_i, which is delegated to $\mathcal{F}_{\mathsf{pake}}$.

We have found that $\Pr[Z_8] = \Pr[Z_9]$. From the preceding games we can conclude that under the SDSH and DDH assumptions,

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

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

- Sim notes down P.

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

- **In Round $\mathbf{R}_1$:** Sim samples $(\mathsf{VK}, \mathsf{SK}) \xleftarrow{\$} \mathsf{OTS.Gen}(1^\lambda)$ 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 msg_2^*:** If $\mathsf{msg}_2^* \neq \mathsf{msg}_2$, Sim finds I/F^κ and sends $(\mathsf{TestPwd}, \mathsf{sid}, \mathsf{client}, I/F^\kappa)$ to $\mathcal{F}_{\mathsf{pake}}$. Otherwise it skips this step and sends $(\mathsf{NewKey}, \mathsf{sid}, \mathsf{client}, 0_{\mathbb{G}})$.
  - **If $\mathcal{F}_{\mathsf{pake}}$ outputs 'correct guess':** Sim 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':** Sim continues the simulation as described in $\mathbf{G}_7$ and submits $(\mathsf{NewKey}, \mathsf{sid}, \mathsf{client}, \mathsf{sk}_{\mathsf{client}})$ to $\mathcal{F}_{\mathsf{pake}}$.

- **In Round $\mathbf{R}_3$ (Client):** Sim selects $x_1, y_1, z_1, w_1 \xleftarrow{\$} \mathbb{Z}_q$ and sets

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

  and sends $\mathsf{msg}_3 = (K, \mathsf{sig})$.

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

- **In Round $\mathbf{R}_2$:** Sim selects $x_2, y_2, z_2, w_2, r_2 \xleftarrow{\$} \mathbb{Z}_q$ and sets

$$\begin{aligned} \alpha' &:= \mathsf{HF}_{\mathsf{hk}}^{\lambda,\Gamma}(\mathsf{sid}, \mathsf{VK}, A, B, C) \\ E &:= g_1^{x_2} g_2^{y_2} h^{z_2} (cd^{\alpha'})^{w_2} \\ F &:= g_1^{r_2} \\ G &:= g_2^{r_2} \\ I &:= h^{r_2} \cdot g^{N+1} \\ \beta &:= \mathsf{HF}_{\mathsf{hk}}^{\lambda,\Gamma}(\mathsf{msg}_1, \mathsf{sid}, E, F, G, I) \\ J &:= (cd^{\beta})^{r_2} \end{aligned}$$

- **On receiving msg_3^*:** If $\mathsf{msg}_1^* \neq \mathsf{msg}_1$, Sim finds C/A^κ and sends $(\mathsf{TestPwd}, \mathsf{sid}, \mathsf{client}, C/A^\kappa)$ to $\mathcal{F}_{\mathsf{pake}}$. Otherwise it skips this step and sends $(\mathsf{NewKey}, \mathsf{sid}, \mathsf{client}, 0_{\mathbb{G}})$.
  - **If $\mathcal{F}_{\mathsf{pake}}$ outputs 'correct guess':** Sim continues the simulation as described in $\mathbf{G}_2$ and submits $(\mathsf{NewKey}, \mathsf{sid}, \mathsf{client}, \mathsf{sk}_{\mathsf{server}})$ to $\mathcal{F}_{\mathsf{pake}}$.
  - **If $\mathcal{F}_{\mathsf{pake}}$ outputs 'wrong guess':** Sim continues the simulation as described in $\mathbf{G}_4$ and submits $(\mathsf{NewKey}, \mathsf{sid}, \mathsf{client}, \mathsf{sk}_{\mathsf{server}})$ to $\mathcal{F}_{\mathsf{pake}}$.

Figure 6: The Simulator for Theorem 4.1.

References

[ABB+20] Michel Abdalla, Manuel Barbosa, Tatiana Bradley, Stanislaw Jarecki, Jonathan Katz, and Jiayu Xu. Universally composable relaxed password authenticated key exchange. pages 278–307, 2020.

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

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

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

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

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

[BDZ03] Feng Bao, Robert H. Deng, and Huafei Zhu. Variations of Diffie-Hellman problem. 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. pages 121–151, 2020.

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

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

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

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

[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. pages 620–635, 2024.

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

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

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

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

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

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

[HL19] Björn Haase and Benoît Labrique. AuCPace: Efficient verifier-based PAKE protocol tailored for the IIoT. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 1–48, 2019.

[Jab97] David P. Jablon. Extended password key exchange protocols immune to dictionary attacks. In *6th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE 1997)*, pages 248–255, Cambridge, MA, USA, June 18–20, 1997. IEEE Computer Society.

[Jar22] Stanislaw Jarecki. Password authenticated key exchange: Protocols and security models. pages 213–255. Wiley, 2022.

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

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

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

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

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

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

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

[PS04] Manoj Prabhakaran and Amit Sahai. New notions of security: Achieving universal composability without trusted setup. 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). pages 714–743, 2023.

[Sho01] Victor Shoup. A proposal for an ISO standard for public key encryption. Cryptology ePrint Archive, Paper 2001/112, 2001.

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 $\mathsf{AdvDRCCA}(\lambda)$, 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* $\lambda \in \mathbb{Z}_{\geq 0}$*,*

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

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

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

Theorem A.1. *If the DDH and the DSDH assumptions hold 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 decryption 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 $(u_1^*, \mathsf{aux}_1[0])$ 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 $(\psi^*, \mathsf{aux}_1) := ((u_1^*, u_2^*, e^*, v^*), (g_1^s, g_2^{s_1}, (h^r \cdot m_b)^r, (cd^\alpha)^{s_2}))$.

The proof essentially follows the same as that of the previous game, and can be omitted. The same can be applied to the SDH tuple $(g, h^r \cdot m_b, (h^r \cdot m_b)^r)$, which we also fix in this game (in this modified version, the adversary simply adjoins an m_b factor to the (g, h^r, h^{r^2}) tuple). Jiayu: What does the last sentence mean?

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

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

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 $\mathbf{D4}'$ 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

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

Jiayu: Why does the second line of the equation hold? 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.