# On the UC-(In)Security of PAKE Protocols in the Random Oracle Model

Naman Kumar*
Oregon State University

Jiayu Xu†
Oregon State University

November 19, 2024

**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 seen increasing interests in recent years, and are interesting from both theoretical and practical perspectives: with the standardization process by the IETF in 2019–20, 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, whose construction involves a number of technically 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, SPAKE2 [AP05], and Jiayu: FILL IN. These protocols are generally extremely efficient but heavily rely on the Random Oracle Model (ROM); in some cases [BM92], 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 [**?**] 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 has seen a number of followup works including generalization [GL03] and simplification [ABP15], and extension to the threshold setting [**?**]. Subsequent works following this paradigm include [**?**] Aside from not requiring an RO, SPHF-based PAKE protocols also have reasonable efficiency; for example, the computational cost of the KOY protocol is roughly 4 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 session. 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 [**?**].

The UC definitions have been proven to imply the (standard) game-based definition ([CHK+05, Appendix A] and [ABB+20, Section 5]). The reverse 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. 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 integer $x$ and random group element $M$ in the CRS), 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 protocol session ends*. This means that SPAKE2 realizes a weaker "lazy-extraction" version of UC PAKE functionality, where 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. 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 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 SPHF-based PAKE protocols [KOY01], the simulator can extract the password guess but cannot compute the session key afterwards, because the password extraction happens "too late".
   To make SPHF-based PAKE protocols UC-secure (without introducing the ROM), [CHK+05] modifies KOY by adding a "pre-commitment" to the password to allow for early extraction, as well as a zero-knowledge 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. The most frustrating point is that the sole purpose of this ZK proof — which destroys the efficiency of KOY[1] — seems to be allowing the simulation to complete, and it is not even clear what exact attack this failure of UC-security implies. Indeed, the trick of adding a ZK proof has been called "mysterious" [**?**]

[1] KOY requires 4 group exponentiations + 2 multi-exponentiations per party; the protocol in [CHK+05] requires around 30 group exponentiations per party.

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.[2] 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.[3] 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 ZK 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.

**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

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

[3] 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].

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 UC-security 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.

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 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 argument.

### 1.1 Protocol Description

Jiayu: Insert the description of KOY (revised to fit the UC formality) here. Naman: I've made edits, clarified the scheme.

We provide a high-level description of the protocol along with a sketch of the security proof below. Let $\mathbb{G}$ be an algebraic group in which DDH is believed to be hard and define $\mathsf{crs} := (g_1, g_2, h, c, d)$ to be uniformly sampled from $\mathbb{G}^5$; we set this to be the common random string. Note that $\mathsf{crs}$ can be interpreted to be a degenerate Cramer-Shoup public key with an unknown secret key.

- $\mathsf{client}$ begins by generating a pair of keys $(\mathsf{VK}, \mathsf{SK})$ for a one-time signature scheme and encrypts its password $\mathsf{pw}$ with label $\mathsf{VK}$ using the Cramer-Shoup public key embedded in the CRS. 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 $\alpha = H(\mathsf{VK}, A, B, C)$. $\mathsf{client}$ then sends $\mathsf{msg}_1 := (\mathsf{VK}, \mathsf{ct}_1)$ to $\mathsf{server}$. Note that $A$, $B$, $C' = C/\mathsf{pw}$, $D$ are all of the form $g^{r_1}$ where $g$ is some group element that $\mathsf{server}$ can compute, so they also serve as a message in a Diffie–Hellman-like protocol.

- Upon receiving $(\mathsf{VK}, A, B, C, D)$, the $\mathsf{server}$ samples its 'Diffie–Hellman exponents' $(x_2, y_2, z_2, w_2)$, and computes $E = g_1^{x_2} g_2^{y_2} h^{z_2} (cd^{\alpha})^{w_2}$.[4] Furthermore, $\mathsf{server}$ encrypts $\mathsf{pw}$ with label $(\mathsf{msg}_1, E)$ using Cramer–Shoup encryption as done by $\mathsf{client}$ in $\mathsf{msg}_1$. Let $r_2$ be the randomness used in encryption. The resulting ciphertext is $\mathsf{ct}_2 := (F, G, I, J) = (g_1^{r_2}, g_2^{r_2}, h^{r_2} \cdot \mathsf{pw}, (cd^{\beta})^{r_2})$. $\mathsf{server}$ sends $\mathsf{msg}_2 = (E, \mathsf{ct}_2)$ to $\mathsf{client}$.

- Symmetrically, $\mathsf{client}$ upon receiving $(E, \mathsf{ct}_2)$ 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 $\mathsf{st} = \mathsf{Sign}_{\mathsf{SK}}(\mathsf{msg}_1 | \mathsf{msg}_2 | K)$, and sends $\mathsf{msg}_3 = (K, \mathsf{st})$ to the server.

- $\mathsf{server}$ verifies the signature $\mathsf{st}$ and aborts if it is invalid. Otherwise the session key $\mathsf{sk}$ is defined as the product of

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

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

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.[5] Therefore, an adversarial user 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 user/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 $\mathsf{msg}_2$ and $\mathsf{msg}_3$ without modification as long as it passes $\mathsf{msg}_1$. 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 scheme 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 $\mathsf{sk}_S = \mathsf{sk}_U^2$. 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 $\alpha$. In this way if the adversary changes VK while keeping the ciphertext $A|B|C|D$, the ciphertext would become invalid.

## 1.2 Technical Overview

In this section, we provide a high-level explanation of why the KOY protocol is insecure in the UC framework, and how the AGM circumvents the difficulty for the UC simulator.

**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 $\mathsf{sk} = E^{r_1} F^{x_1} G^{y_1} (I/\mathsf{pw})^{z_1} J^{w_1}$. 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 generated 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"

[5] 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 function is defined as $\mathsf{Hash}_{(x_2,y_2,z_2,w_2)}(m) = A^{x_2} B^{y_2} (C/m)^{z_2} D^{w_2}$.

$\ell = \log_{g_1} h$. After receiving the NewSession command from $\mathcal{F}_{\mathsf{pake}}$, $\mathcal{S}$ must simulate client's first message $\mathsf{msg}_1$. Since $\mathcal{S}$ does not know the password, at this point it must (effectively) guess some $\mathsf{pw}'$ at random; that is, in $\mathsf{msg}_1$, $C = h^{r_1} \cdot \mathsf{pw}'$ where $\mathsf{pw}'$ can be no better than a random password sampled from the dictionary. ($C$ is indistinguishable from the correct value due to the security of Cramer–Shoup encryption.) After $\mathcal{Z}$ responds with $\mathsf{msg}_2 = E|F|G|I|J$, since $F = g_1^{r_2}$ and $I = h^{r_2} \cdot \mathsf{pw}$, $\mathcal{S}$ can extract pw as $I/F^{\ell}$. Once this has been done, $\mathcal{S}$ can send a TestPwd command to $\mathcal{F}_{\mathsf{pake}}$ on the correct pw; $\mathcal{F}_{\mathsf{pake}}$ would mark the client session compromised and thus allow $\mathcal{S}$ to choose client's session key sk (which has to be consistent with the session key client computes in the real world).[6] 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 $\mathsf{msg}_2$ and sampled $r_1$ before sending $\mathsf{msg}_1$. However, the problem is that *the password guess* $\mathsf{pw}'$ *that* $\mathcal{S}$ *uses while generating* $\mathsf{msg}_1$ *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 $1/|\mathcal{D}|$).

**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 $\mathsf{msg}_2$. 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 $\mathsf{msg}_1$, and pw. An algebraic $\mathcal{Z}$ must "explain" how $E$ is computed; for now let's ignore $A, B, C, D, \mathsf{pw}$ 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 $\mathsf{pw}'$ with high probability while generating $A, B, C, D$; however, after extracting the correct pw from $\mathsf{msg}_2$, $\mathcal{S}$ 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

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

[6] A TestPwd *must* be run, since we require that $\mathsf{msg}_1$ and $\mathsf{msg}_2$ 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.

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 $\mathsf{pw}'$ is used lies in $h$, so we only need to "correct" $h^{r_1}$ from $C/\mathsf{pw}'$ to $C/\mathsf{pw}$. 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 $\mathsf{msg}_1$. (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 $\mathsf{sk}_U = X_1 X_2$ 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 $r_1$ is the randomness used in $\mathsf{msg}_1$). 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 $\mathsf{pw}'$ 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 sees $E^{r_1}$ for some $E$ of its choice.*

Below we analyze two simple attacks using this strategy:

1. Say $\mathcal{Z}$ chooses $E = A = g_1^{r_1}$; then $E^{r_1} = g_1^{r_1^2}$. If 2-DL is easy in the group[7], 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 $\mathsf{msg}_1$. In fact, it seems that for the KOY protocol to be UC-AGM-secure, we need the Decisional Square Diffie–Hellman assumption (DSDH), which says that given $g^x$, it is hard to distinguish $g^{x^2}$ from random. Note that this implies 2-DL (given $(g^x, Y)$ where $Y$ is either $g^{x^2}$ or random, the DSDH solver can feed $(g^x, Y)$ to the 2-DL solver and distinguish by observing whether the 2-DL solver wins or not).

2. Say $\mathcal{Z}$ chooses $E = c$; then $E^{r_1} = c^{r_1}$. 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:

[7] 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].

(a) $\mathcal{R}$ embeds the challenge ciphertext as $\mathsf{msg}_1$, $\mathsf{client}$'s first message intercepted by $\mathcal{A}$;

(b) When $\mathcal{A}$ sends $\mathsf{msg}_1^*$ to $\mathsf{server}$ and $\mathsf{msg}_2^*$ to $\mathsf{client}$, $\mathcal{R}$ needs to query the decryption oracle to extract the password guesses contained in these two messages;

(c) Finally, if the password guess in $\mathsf{msg}_2^*$ is correct, $\mathcal{R}$ needs to simulate $\mathsf{sk}_{\mathsf{client}}$. (This step is not needed in the game-based proof).[8]

$c^{r_1}$ 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 in Appendix A.

3. Finally, we note that the *only* attacks are those which allow the algebraic adversary to learn some combination of $g_1^{r_1^2}, g_2^{r_1^2}, c^{r_1}, d^{r_1}, (h^{r_1} \cdot m_i)^{r_1}, (cd^{\alpha})^{r_1{}^2}$. All group elements obtained by the environment are some algebraic combination of the above and $h$; however, attacks involving $h$ can be simulated. To see this, consider the case where the adversary merely sets $E = h$, and learns $h^{r_1}$. This hypothetically should allow the adversary to be able to learn if $\mathsf{pw}' = \mathsf{pw}$ and break the scheme. However, this is not the case by way of the session key of the client is determined – note that in this case, the adversary in fact learns $h^{r_1}(\mathsf{pw}/\mathsf{pw}')$, which when used to "decrypt" provides only $\mathsf{pw}$ as output. This happens because the AGM simulator is able to fix the term $C'$ that involves $h$ while simulating the session key, leading us to conclude that the only possible attacks are those as described above.

By inspecting the game-based security proof of the KOY protocol, one can see that the above attacks are essentially the only scenarios where the UC-security might be broken, 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.

# 2 Preliminaries

## 2.1 Assumptions

Let $\mathcal{G}$ be an algorithm that on input $1^\lambda$ uniformly samples from a sequence of group distributions $(S_\lambda)_{\lambda \in \mathbb{Z}_{\geq 0}}$ of group distributions of the form $\Gamma[\hat{\mathbb{G}}, \mathbb{G}, g, q] \in [S_\lambda]$, where $\hat{\mathbb{G}}$ is a finite multiplicative abelian group, $\mathbb{G}$ is a prime-order subgroup, $g$ is a generator of $\mathbb{G}$ and $q$ is the order of $\mathbb{G}$. We assume that group operations in $\mathbb{G}$ can be done in (expected) polynomial time in $\mathbb{G}$, including exponentiation, selecting a random group element, and membership.

Note that a random group element can be chosen by selecting $x \xleftarrow{\$} \mathbb{Z}_q$ and computing $g^x$. 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 $x \xleftarrow{\$} \bar{\mathbb{G}}$, then we obtain a random generator.

### 2.1.1 The DDH Assumption

Let $\mathcal{G}$ be as above. We define for each $\lambda \in \mathbb{Z}_{\geq 0}$ and for all $\Gamma[\hat{\mathbb{G}}, \mathbb{G}, g, q] \in [S_\lambda]$ the sets

$$\mathsf{RandDH}_{\lambda,\Gamma} := \{(g, g^x, g^y, g^z) : x, y \xleftarrow{\$} \mathbb{Z}_q^*, z \xleftarrow{\$} \mathbb{Z}_q\}$$

[8] We omit the remaining steps which are to simulate $\mathsf{msg}_3$, and on $\mathsf{msg}_3^*$ simulate $\mathsf{server}$'s session key $\mathsf{sk}_{\mathsf{server}}$, as they are inconsequential to our main point.

and

$$\mathsf{DH}_{\lambda,\Gamma} := \{(g, g^x, g^y, g^{xy}) : x, y \xleftarrow{\$} \mathbb{Z}_q^*\}.$$

Where $\mathsf{DH}_{\lambda,\Gamma}$ is the set of Diffie-Hellman Triples. For any probabilistic polynomial-time distinguishing algorithm $\mathcal{A}$ and for all $\lambda \in \mathbb{Z}_{\geq 0}$, we define the DDH advantage of $\mathcal{A}$ against $\mathcal{G}$ at $\lambda$ given $\Gamma$ as

$$\mathsf{AdvDDH}_{\mathcal{G},\mathcal{A}}(\lambda|\Gamma) = \left|\Pr[\mathcal{A}(1^\lambda, \Gamma, \rho) = 1 : \rho \xleftarrow{\$} \mathsf{RandDH}_{\lambda,\Gamma}] - \Pr[\mathcal{A}(1^\lambda, \Gamma, \rho) = 1 : \rho \xleftarrow{\$} \mathsf{DH}_{\lambda,\Gamma}]\right|$$

The Decisional Diffie-Hellman (DDH) assumption for $\mathcal{G}$ states that for every probabilistic, polynomial-time algorithm $\mathcal{A}$, there exists some negligible function $\mathsf{negl}$ such that for all $\lambda \in \mathbb{Z}_{\geq 0}$,

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

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

### 2.1.2 The SDH Assumption

Let $\mathcal{G}$ be as above. We define for each $\lambda \in \mathbb{Z}_{\geq 0}$ and for all $\Gamma[\hat{\mathbb{G}}, \mathbb{G}, g, q] \in [S_\lambda]$ the sets

$$\mathsf{RandSDH}_{\lambda,\Gamma} := \{(g, g^x, g^y) : x, y \xleftarrow{\$} \mathbb{Z}_q^*\}$$

and

$$\mathsf{SDH}_{\lambda,\Gamma} := \{(g, g^x, g^{x^2}) : x \xleftarrow{\$} \mathbb{Z}_q^*\}.$$

Where $\mathsf{SDH}_{\lambda,\Gamma}$ is the set of Square Diffie-Hellman Triples. For any probabilistic polynomial-time distinguishing algorithm $\mathcal{A}$ and for all $\lambda \in \mathbb{Z}_{\geq 0}$, we define the SDH advantage of $\mathcal{A}$ against $\mathcal{G}$ at $\lambda$ given $\Gamma$ as

$$\mathsf{AdvSDH}_{\mathcal{G},\mathcal{A}}(\lambda|\Gamma) = \left|\Pr[\mathcal{A}(1^\lambda, \Gamma, \rho) = 1 : \rho \xleftarrow{\$} \mathsf{RandSDH}_{\lambda,\Gamma}] - \Pr[\mathcal{A}(1^\lambda, \Gamma, \rho) = 1 : \rho \xleftarrow{\$} \mathsf{SDH}_{\lambda,\Gamma}]\right|$$

The Square Diffie-Hellman (DDH) assumption for $\mathcal{G}$ states that for every probabilistic, polynomial-time algorithm $\mathcal{A}$, there exists some negligible function $\mathsf{negl}$ such that for all $\lambda \in \mathbb{Z}_{\geq 0}$,

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

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

## 2.2 Cramer-Shoup Encryption Scheme

## 2.3 UC-PAKE

We define the UC-PAKE functionality below.

# 3 Proof of UC-Insecurity

## 3.1 Formal Description of the Protocol

We provide a formal description of the PAKE protocol of KOY in the figure below.

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

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

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 $1^\lambda$ for $\lambda \in \mathbb{Z}_{\geq 0}$, 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 $\mathsf{label}$, compute

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

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

**Decryption:** Given $1^\lambda$ for $\lambda \in \mathbb{Z}_{\geq 0}$, 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 $\psi$ and a label $\mathsf{label}$, do the following.

**D1**: Parse $\psi$ as a 4-tuple $(u_1, u_2, e, v) \in \mathbb{G}^4$, output $\mathsf{reject}$ and halt if $\psi$ is not of this form.
**D2**: Test if $u_1, u_2$ and $e$ belong to $\mathbb{G}$; output $\mathsf{reject}$ and halt if this is not the case.
**D3**: Compute $v' \leftarrow \mathsf{HF}_{\mathsf{hk}}^{\lambda,\Gamma}(\mathsf{label}, u_1, u_2, e)$.
**D4**: Test if $v = u_1^{x_1+y_1\alpha} u_2^{x_2+y_2\alpha}$; output $\mathsf{reject}$ and halt if this is not the case.
**D5**: Compute $e' = u_1^{z_1} u_2^{z_2}$.
**D6**: Compute $m \leftarrow e \cdot e'^{-1}$ and output $m$.

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

- On input $(\mathsf{NewSession}, sid, \mathsf{P}, \mathsf{P}', \mathsf{pw}, \mathsf{role})$ from $\mathsf{P}$, send $(\mathsf{NewSession}, sid, \mathsf{P}, \mathsf{P}', \mathsf{role})$ to $\mathsf{Sim}$. 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 $\mathsf{Fresh}$.
- On $(\mathsf{TestPwd}, sid, \mathsf{P}, \mathsf{pw}^*)$ from $\mathsf{Sim}$, if there is a record $\langle \mathsf{P}, \mathsf{P}', \mathsf{pw} \rangle$ marked $\mathsf{Fresh}$, then do:
  - If $\mathsf{pw}^* = \mathsf{pw}$, then mark the record $\mathsf{compromised}$ and send "correct guess" to $\mathsf{Sim}$.
  - If $\mathsf{pw}^* \neq \mathsf{pw}$, then mark the record $\mathsf{interrupted}$ and send "wrong guess" to $\mathsf{Sim}$.
- On $(\mathsf{NewKey}, sid, \mathsf{P}, K^* \in \{0,1\}^\lambda)$ from $\mathsf{Sim}$, if there is a record $\langle \mathsf{P}, \mathsf{P}', \mathsf{pw} \rangle$, and this is the first $\mathsf{NewKey}$ message for $sid$ and $\mathsf{P}$, then output $(sid, K)$ to $\mathsf{P}$, where $K$ is defined as follows:
  - If the record is $\mathsf{compromised}$, or either $\mathsf{P}$ or $\mathsf{P}'$ is corrupted, then set $K := K^*$.
  - If the record is $\mathsf{Fresh}$, a key $(sid, K')$ has been output to $\mathsf{P}'$, at which time there was a record $\langle \mathsf{P}', \mathsf{P}, \mathsf{pw} \rangle$ marked $\mathsf{Fresh}$, then set $K := K'$.
  - Otherwise sample $K \leftarrow \{0,1\}^\lambda$.

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

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

### 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 $\mathsf{client}$ and $\mathsf{server}$ 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 $\mathcal{S}$.
2. $\mathcal{R}$ waits to receive $\mathsf{msg}_1 = \mathsf{sid}|\mathsf{VK}|A|B|C|D$ in response (as the first message from $\mathsf{client}$ to

**Initialization**: Let $\Gamma[\hat{\mathbb{G}}, \mathbb{G}, g, q] \xleftarrow{\$} [S_\lambda]$. Set $g_1 \leftarrow g$ and set $g_2, c, d, h \xleftarrow{\$} \overline{\mathbb{G}}$. Furthermore, $\mathsf{hk} \xleftarrow{\$} \mathsf{HF.Keyspace}_{\lambda,\Gamma}$. Output $\mathsf{crs} = (\Gamma, g_1, g_2, h, c, d, \mathsf{hk})$.

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

1. **Initiation.** Party $\mathsf{P}$ sends the message $(\mathsf{NewSession}, \mathsf{sid}, \mathsf{P}, \mathsf{P}', \mathsf{role})$. Party $\mathsf{P}'$ replies with $(\mathsf{NewSession}, \mathsf{sid}, \mathsf{P}', \mathsf{P}, \mathsf{role})$.

2. **Round $\mathbf{R}_1$.** client gets $(\mathsf{VK}, \mathsf{SK}) \xleftarrow{\$} \mathsf{OTS.Gen}(1^\lambda)$ and selects $r_1 \xleftarrow{\$} \mathbb{Z}_q$. It sets

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

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

3. **Round $\mathbf{R}_2$.** server selects $x_2, y_2, z_2, w_2, r_2 \xleftarrow{\$} \mathbb{Z}_q$ and sets

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

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

4. **Round $\mathbf{R}_3$ (Client).** client 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 $\mathsf{msg}_3 = (K, \mathsf{sig})$. It finds $I' = I/\mathsf{pw}_{\mathsf{client}}$ and outputs

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

5. **Round $\mathbf{R}_3$ (Server).** Server receives $\mathsf{msg}_3$ and checks if $\mathsf{Vrfy}_{\mathsf{VK}}(\mathsf{msg}_1 \| \mathsf{msg}_2 \| K, \mathsf{sig}) = 1$. If not, it outputs $\mathsf{sk}_{\mathsf{server}} = \mathsf{NULL}$. Else, it finds $C' = C/\mathsf{pw}_{\mathsf{client}}$ and outputs

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

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 $\mathsf{client}$.

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 $\mathsf{client}$.

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

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 $\mathcal{Z}$.

$\mathsf{server}$). 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 $\mathsf{client}$ to $\mathsf{server}$) and $(\mathsf{sid}, \mathsf{sk})$ from $\mathcal{S}$ (as $\mathsf{client}$'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 $\mathsf{pw}$ 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

We will show the following theorem.

**Theorem 4.1.** *The protocol of Fig. 3 UC-realizes the $\mathcal{F}_{\mathsf{pake}}$ functionality in the Algebraic Group Model under the SDH and DDH Assumptions.*

## 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 advantage of the adversary in game $\mathbf{G}_0$ (which corresponds to the real-world attack) is at most negligibly greater than that of the adversary in game $\mathbf{G}_9$, which corresponds to the ideal world.

Let $Z_i$ denote the event that $\mathcal{Z}$ outputs 1 in game $G_i$. We need to show that $Z_0 \cong Z_9$, which corresponds to the fact that the real world attack is indistinguishable from the ideal world attack.

We refer to the system of $\mathsf{P}_0$ and $\mathsf{P}_1$ as the *challenger*. The challenger plays the role of $\mathsf{client}$ and $\mathsf{server}$ and carries out all interactions with the adversary (and hence, the environment). We denote each password as a group element $\mathsf{pw} = g^{\mathsf{pw}'}$ where $\mathsf{pw}$ is an element of the password dictionary $\{1, \ldots, N\}$. We assume that the size of the password dictionary is small, ie. $N < q$.

Let $\mathsf{msg}_i$ denote the $i$th message output by the challenger. This message is relayed to the adversary, and the (algebraic) adversary responds with a message $\mathsf{msg}_i^*$ along with a message $\mathsf{msg}_i^{'*}$ which is an algebraic representation of the elements it has seen so far.

## 4.2 Overview

The broad strokes of our proof are as below.

1. In game $\mathbf{G}_0'$, the experiment is aborted if any of the following events happen.
   (a) A $\mathsf{VK}$ is reused. In this case, the challenger detects this in the message that is output, sends an $\mathsf{abort}$ and ends the protocol.
   (b) The server generates $\mathsf{msg}_2$ twice for different $\mathsf{sid}$s. The challenger detects this in the message that is output, sends an $\mathsf{abort}$ and ends the protocol.
   (c) The challenger detects at some point that the adversary has been able to produce a new signed message for some honestly-generated $\mathsf{VK}$ without knowing the corresponding signing key (which it can't, because the $\mathsf{VK}$ is honestly generated – if it was the one who generated that $\mathsf{VK}$, the adversary is allowed to do whatever it wants with it).

(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 $\mathsf{abort}$ 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 functions and the one-time signature scheme.

2. In game $\mathbf{G}_1$, the challenger changes the way that the $\mathsf{crs}$ is generated. In this case, the $\mathsf{crs}$ is now generated according to the techniques laid out in the Cramer-Shoup $\mathsf{KeyGen}$ procedure. The distribution of generated values is information-theoretically identical; the only difference is that now the challenger has planted trapdoors into the description of the $\mathsf{crs}$, and it knows the discrete logs of $h, c$ and $d$. The simulator is going to (eventually) use this to extract a password guess.

3. In game $\mathbf{G}_2$, the challenger considers the messages $\mathsf{msg}_i^*$ received from the adversary, and uses the parameters in $\mathbf{G}_1$ to extract a password guess. If the password guess is correct, it extracts the computed session key in the compromised protocol and sets it to be $\mathsf{sk}_i$. Note that the change is purely conceptual; the game is indistinguishable from the previous one. The extraction of $\mathsf{sk}_i$ requires us to use the algebraic exponents delivered by the adversary; this part of the proof requires the AGM.

4. In game $\mathbf{G}_3$, the challenger deals with incoming $\mathsf{msg}_3^*$ queries. It checks the $\mathsf{sid}$ of the message and in case that $\mathsf{msg}_1 = \mathsf{msg}_1^*$, it sets $\mathsf{sk}_{\mathsf{server}} := \mathsf{sk}_{\mathsf{client}}$ for the same $\mathsf{sid}$, which was already previously computed (at the time that $\mathsf{msg}_3$ was output). Note that in this case $\mathsf{msg}_3$ *has* to be output by the client – the only situation in which this doesn't happen is if the environment was able to forge some $(K, \mathsf{sig})$, but $\mathbf{G}_0'$ necessitates abort in the case that this occurs. Thus, the game is valid, and furthermore is indistinguishable because this is, again, only a conceptual change.

5. In game $\mathbf{G}_4$, the challenger again deals with incoming $\mathsf{msg}_3^*$ queries. It again checks the $\mathsf{sid}$ of the message and in the case that $\mathsf{msg}_1 \neq \mathsf{msg}_1^*$ and furthermore the message is an invalid Cramer-Shoup ciphertext of $\mathsf{pw}_{\mathsf{client}}$, the challenger outputs a session key $\mathsf{sk}_{\mathsf{server}}$ *at random*. The proof of indistinguishability proceeds from an information-theoretic argument.

6. In game $\mathbf{G}_5$, the challenger replaces the server's Cramer-Shoup encryption of $\mathsf{pw}_{\mathsf{client}}$ with an encryption of some $g^k \notin \mathcal{PW}$. The proof of indistinguishability follows from the fact that any environment which can differentiate between the two games can also be used as a subroutine to break the CCA-security of Cramer-Shoup.

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

8. In game $\mathbf{G}_7$, the challenger considers $\mathsf{msg}_2^*$ again, and the case in which $\mathsf{msg}_2^* \neq \mathsf{msg}_2$ but the message does not contain a valid password guess, it sets $\mathsf{sk}_{\mathsf{client}}$ to be random as well. The proof is again similar.

9. In game $\mathbf{G}_8$, the challenger replaces the client's Cramer-Shoup encryption of $\mathsf{pw}_{\mathsf{client}}$ with an encryption of some $g^k \notin \mathcal{PW}$. The proof of indistinguishability follows from the DR-CCA security of Cramer-Shoup, which is discussed in Appendix A.

10. Game $\mathbf{G}_9$ is the ideal functionality. We argue that the structure of $\mathbf{G}_8$ can be interpreted as the ideal functionality with no security loss.

### 4.3 Proof of UC-Security

We now provide a formal description of the proof.

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

**Game $\mathbf{G}_0'$.** In this game, the challenger interacts with the adversary as before, except we introduce an additional abort clause. If this clause is activated, then the challenger sends a message $\mathsf{abort}$ and ends the protocol. The abort clause is activated in the case of the following situations:

1. The verification key $\mathsf{VK}$ is used more than once by the system. This encompasses the case when the client's generation algorithm outputs $\mathsf{VK}$ twice for two separate instantiations in two separate sessions.

2. The server generates a $\mathsf{msg}_2$ which is identical to a different $\mathsf{msg}_2$ generated in a separate session.

3. Consider $(\mathsf{SK}, \mathsf{VK})$ to be honest outputs of $\mathsf{OTS.KeyGen}(1^\lambda)$ by the client. Then if the adversary is able to output some $\mathsf{sig}'$ which was *not* generated by the challenger using the signing algorithm, and $\mathsf{Vrfy}_{\mathsf{VK}}(\mathsf{sig}') = 1$.

4. A collision occurs in the hash function.

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

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

(4) occurs with negligible probability by the collision-resistance of the hash function $\mathsf{HF}$.

Case (1) and (3) occur with negligible probability by fact that $\mathsf{OTS}$ 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 $\mathsf{crs}$ is sampled. In particular, it uses the following 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 $\Gamma[\hat{\mathbb{G}}, \mathbb{G}, g, q] \xleftarrow{\$} [S_\lambda]$. Set $g_1 \leftarrow g$, $g_2 \xleftarrow{\$} \overline{\mathbb{G}}$. Choose

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

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, $\mathsf{hk} \xleftarrow{\$} \mathsf{HF.Keyspace}_{\lambda,\Gamma}$. 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 $\mathsf{crs}$ 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 $\overline{\mathbb{G}}$. Thus, 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*.
2. Consider $\mathsf{msg}_1^*$. If either $A^{\chi_1+\alpha\zeta_1}B^{\chi_2+\alpha\zeta_2} \neq D$ or $C/\mathsf{pw}_{\mathsf{client}} \neq A^{\kappa}$, then $\mathsf{msg}_1$ is *invalid*, otherwise it is *valid*.
3. Consider $\mathsf{msg}_2^*$. 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 $\mathsf{msg}_1$ is *invalid*, otherwise it is *valid*.

**Game $\mathbf{G}_2$.** In this game, the challenger makes two changes. After this game, the password will no longer be used in computing the session key, except implicitly.

When the challenger receives $\mathsf{msg}_2^*$, it examines it. In the case that $\mathsf{msg}_2^*$ is adversarially-generated and valid, it considers the adversary's algebraic exponents $(x_2, y_2, z_2, w_2, r_2)$, which were provided along with $\mathsf{msg}_2^*$. 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 $\mathsf{msg}_3^*$ exactly as it does in $\mathbf{G}_1$.

When the challenger receives $\mathsf{msg}_3^*$, it examines $\mathsf{msg}_1^*$ for the corresponding $\mathsf{sid}$. If $\mathsf{msg}_1^*$ is adversarially generated and valid, it considers and the adversary's algebraic exponents output with $\mathsf{msg}_3^*$ and $\mathsf{msg}_1^{'*}$ as $(x_1, y_1, z_1, w_1, r_1)$. 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.** *We have*

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

*Proof.* It is clear that the distribution on the adversary's (and environment'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 $\mathsf{sk}_i$ between $\mathbf{G}_1$ and $\mathbf{G}_2$, and the proposition follows. □

**Game $\mathbf{G}_3$.** In this game the challenger modifies how it deals with an incoming $\mathsf{msg}_3^*$. Upon receiving this message, it checks the corresponding $\mathsf{msg}_1^*$ of $\mathsf{sid}$. If $\mathsf{msg}_1^*$ is not adversarially-generated, then it finds the unique $\mathsf{sk}_{\mathsf{client}}$ for $\mathsf{sid}$ and sets $\mathsf{sk}_{\mathsf{server}} := \mathsf{sk}_{\mathsf{client}}$. In any other case it proceeds as in $\mathbf{G}_2$.

At first it is not clear why such a corresponding $\mathsf{sk}_{\mathsf{client}}$ exists. However, this follows from the fact that the protocol did not $\mathsf{abort}$. Hence, there is some unique $\mathsf{sid}$ for which the $\mathsf{VK}$ corresponding to $\mathsf{msg}_3^*$ was used, and furthermore this is the same $\mathsf{sid}$ corresponding to $\mathsf{msg}_3^*$, because a case of successful verification if the two $\mathsf{sid}$s were not equal would mean a signature was forged. Both these situations are prohibited by the $\mathsf{abort}$ clause introduced in game $\mathbf{G}_0'$.

Note that in $\mathbf{G}_2$ it is true that in the case of honestly generated $\mathsf{msg}_1$, $\mathsf{sk}_{\mathsf{client}} = \mathsf{sk}_{\mathsf{server}}$. 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 $\mathsf{msg}_3^*$ with a corresponding $\mathsf{msg}_1^*$ that is adversarially-generated but invalid. If this is indeed 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.** *We have*

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

*Proof.* We will proceed via an information-theoretic argument. Note that $\mathsf{msg}_1^*$ is adversarially-generated and invalid. Hence, there are two possibilities: either $A^{\chi_1+\alpha\zeta_1}B^{\chi_2+\alpha\zeta_2} \neq D$, or $C/\mathsf{pw}_{\mathsf{client}} \neq A^{\kappa}$. Consider now the random variables $E$ and $\mathsf{sk}_{\mathsf{server}}$, which are determined completely by the choices of $(x, y, z, w)$ and $(A, B, C, D, K)$. We can rewrite these as the following linear system, in which the logarithms are taken over base $g$:

$$\begin{pmatrix} \log E \\ \log \mathsf{sk}_{\mathsf{server}} - r \log K \end{pmatrix} = \begin{pmatrix} 1 & \log g_2 & \log h & \log cd^{\alpha} \\ \log A & \log B & \log\left(\frac{C}{\mathsf{pw}_{\mathsf{client}}}\right) & \log D \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ w \end{pmatrix} \tag{1}$$

It is easy to see that these equations are linearly independent. 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 $C/\mathsf{pw}_{\mathsf{client}} \neq A^{\kappa}$, then the independence is trivial. Then for any fixed values of $\mu, \nu$, 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 $\mathsf{sk}_{\mathsf{server}}$ appears uniform to the adversary in $\mathbf{G}_3$, which is the same as $\mathbf{G}_4$. □

**Game $\mathbf{G}_5$.** In this game the challenger modifies how it determines $\mathsf{msg}_2$. On receiving $\mathsf{msg}_1^*$, instead of setting $I := h^r \cdot g^{\mathsf{pw}'_{\mathsf{client}}}$, it sets $I := h^r g^{N+1}$ which is not a valid $\mathsf{pw}$ as determined by the password dictionary.

**Lemma 4.3.** *Under the SDSH assumption, we have*

$$|\Pr[Z_4] - \Pr[Z_5]| \leq \mathsf{negl}(\lambda).$$

*Proof.* We show that any environment $\mathcal{Z}$ which can differentiate between $\mathbf{G}_4$ and $\mathbf{G}_5$ can be used as a subroutine to build an adversary $\mathcal{A}$ which can break the DR-CCA-security game (Fig. 7) for Cramer-Shoup with labels (Fig. 1) by successfully distinguishing between target ciphertexts.

We provide a description of $\mathcal{A}$ as follows. $\mathcal{A}$ 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 for the challenger and a dummy adversary. This simulation proceeds as in $\mathbf{G}_4$, with the following difference:

- On receiving a message $\mathsf{msg}_1^*$, $\mathcal{A}$ chooses $m_0 := (\mathsf{pw}_{\mathsf{client}})$ (which was provided by $Z$) and $m_1 := g^{N+1}$ as messages for the encryption oracle. It receives in return a ciphertext $\psi^*$, and sets $(F, G, I, J) := \psi^*$.
- On receiving an adversarially generated message $\mathsf{msg}_2^*$, 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 $\mathsf{msg}_3^*$, the challenger again needs to determine whether $\mathsf{msg}_1^*$ 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 $K$ delivered by the adversary 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 $\mathsf{msg}_1^*$ 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 $\mathsf{aux}$. It then releases $\mathsf{sk}_{\mathsf{server}}$ to $\mathcal{Z}$.

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

It is clear that the above simulation corresponds to a correct simulation of $\mathbf{G}_4$ if the returned ciphertext $\psi^*$ is an encryption of $m_0$ while it corresponds to a simulation of $\mathbf{G}_5$ if the returned ciphertext is a simulation of $m_1$. It follows that

$$|\Pr[Z_4] - \Pr[Z_5]| \leq \mathsf{AdvDRCCA}_{\mathcal{A}}(\lambda) \leq \mathsf{negl}(\lambda).$$

□

**Game $\mathbf{G}_6$.** In this game the challenger deals with incoming message $\mathsf{msg}_2^*$. If $\mathsf{msg}_2^*$ is honest, then it assigns the client's session key $\mathsf{sk}_{\mathsf{client}}$ uniformly chosen in $\mathbb{G}$.

The proof of indistinguishability of this game is closely related to that of game $\mathbf{G}_4$. It is true here that 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 family similar to Eq. (1) follows, and the proof of linear independence is also nearly identical. We can conclude that the session key is uniform in $\mathbf{G}_5$, and equivalence follows immediately:

$$\Pr[Z_5] = \Pr[Z_6].$$

**Game $\mathbf{G}_7$.** In this game the challenger deals with incoming message $\mathsf{msg}_2^*$. If $\mathsf{msg}_2^*$ is adversarially-generated but invalid, then it assigns the client's session key $\mathsf{sk}_{\mathsf{client}}$ uniformly chosen in $\mathbb{G}$.

Note that it is true here as well that either $F^{\chi_1+\beta\zeta_1} G^{\chi_2+\beta\zeta_2} \neq J$ or $I/\mathsf{pw}_{\mathsf{client}} \neq F^{\kappa}$. The argument is identical to $\mathbf{G}_6$, and we get

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

**Game $\mathbf{G}_8$.** 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 $\mathbf{G}_5$, setting it to be an invalid password.

**Lemma 4.4.** *Under the DSDH and the DDH assumptions,*

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

*Proof.* Like the proof of $\mathbf{G}_5$, we will show that any environment $\mathcal{Z}$ which can differentiate between the two games can be used to build an adversary which breaks the DR-CCA game for Cramer–Shoup with labels.

The description of the adversary $\mathcal{A}$ is as follows. $\mathcal{A}$ 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 for the challenger and a dummy adversary. This simulation proceeds as in $\mathbf{G}_7$, with the following difference:

- On receiving a message $\mathsf{NewSession}$, $\mathcal{A}$ chooses $m_0 := (\mathsf{pw}_{\mathsf{client}})$ (which was provided by $Z$) and $m_1 := g^{N+1}$ as messages for the encryption oracle. It receives in return a ciphertext $\psi^*$, and sets $(A, B, C, D) := \psi^*$.
- On receiving an adversarially generated message $\mathsf{msg}_2^*$, 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 $\mathsf{msg}_3^*$, the challenger again needs to determine whether $\mathsf{msg}_1^*$ 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 $\mathsf{msg}_2^{\prime *}$ 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 $\mathsf{msg}_2^*$ 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 $\mathsf{aux}$. 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 $\mathsf{sk}_i$, 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** $\mathcal{F}_{\mathsf{pake}}$:

- $\mathsf{Sim}$ notes down $\mathsf{P}$.

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

- **In Round $\mathbf{R}_1$:** $\mathsf{Sim}$ 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 $\mathsf{msg}_2^*$:** If $\mathsf{msg}_2^* \neq \mathsf{msg}_2$, $\mathsf{Sim}$ finds $I/F^\kappa$ 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':** $\mathsf{Sim}$ 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':** $\mathsf{Sim}$ 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):** $\mathsf{Sim}$ 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 $\mathbf{R}_2$:** $\mathsf{Sim}$ 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 $\mathsf{msg}_3^*$:** If $\mathsf{msg}_1^* \neq \mathsf{msg}_1$, $\mathsf{Sim}$ finds $C/A^\kappa$ 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':** $\mathsf{Sim}$ 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':** $\mathsf{Sim}$ 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.

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

[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.

[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.

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

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

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

# 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 Cryptosystem. 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 $(\mathsf{PK}, \mathsf{SK}) \leftarrow \mathsf{Enc.KeyGen}$ and responds with $\mathsf{PK}$.

**Stage 2:** The adversary makes a sequence of calls to a decryption oracle. For each decryption oracle query, the adversary submits a ciphertext $\psi$, and the decryption oracle responds with $\mathsf{PKE.dec}(1^\lambda, \mathsf{SK}, \psi)$.

**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

$$\sigma \xleftarrow{\$} \{0,1\}; \psi^* \xleftarrow{\$} \mathsf{PKE.Enc}(1^\lambda, \mathsf{PKE}, m_\sigma)$$

and responds with the target ciphertext $\sigma^*$.

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

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

**Stage 5′:** The adversary submits a $\mathsf{halt}$ statement to the decryption oracle. 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 $\hat{\sigma} \in \{0,1\}$.

Figure 7: The Delayed-Reveal-CCA game. The difference in stage 5 is highlighted in blue, which represents the regular CCA-security game.

We define the delayed-reveal-CCA advantage with auxiliary information $\mathsf{aux}$ of the adversary $\mathcal{A}$ against $\mathsf{PKE}$ at $\lambda$, denoted as $\mathsf{AdvDRCCA}(\lambda)$ to be $|\Pr[\sigma = \hat{\sigma}] - 1/2|$ in the above attack game in Fig. 7 with corresponding value of $\mathsf{aux}$. Informally, we say that a public-key cryptosystem is delayed-reveal-CCA secure with auxiliary information $\mathsf{aux}$ if the probability that $\sigma = \hat{\sigma}$ is negligible, ie. the adversary is able to correctly guess which message was encrypted with no more than negligible probability. This leads to the formal definition as below.

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

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

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

**Theorem A.1.** *If the DDH and the SDH assumptions hold for $\mathbb{G}$ and the hash function $H$ is Target Collision-Resistant, then the Cramer-Shoup Encryption Scheme with Labels (Fig. 1) is delayed-reveal-CCA secure with auxiliary information* $\mathsf{aux} = (x_1, x_2, y_1, y_2, g_1^{r^2}, g_2^{r^2}, (h^r \cdot m_i)^r, (cd^\alpha)^{r^2})$.

## 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 CCA-security of the Cramer-Shoup encryption scheme outlined in Section 6.2 of [CS03]. 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 $\sigma$. In each game, $\sigma$ 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 $\hat{\sigma} = \sigma$. 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 $\sigma$, 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 $|\Pr[T_i] - 1/2|$. 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 $\sigma \in \{0, 1\}$ and $r \in \mathbb{Z}_q$ which are generated by the encryption oracle.

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 SDH. It shows that the auxiliary information $\mathsf{aux}_1$ has no 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 $h^r \cdot m_\sigma$, allowing $m_\sigma$ 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 $\psi^*$ does *not* have that particular correlation. The immediate implication is that the adversary cannot with any nontrivial probability modify the target ciphertext $\psi^*$ in such a way that obtains even a different, valid ciphertext, let alone one that provides information about $m_\sigma$. However, the argument of indistinguishability is nontrivial and spreads across more games.

- In game $\mathbf{G}_7$, $m_\sigma$ is replaced by a uniformly random value. We will show that both the advantage of this game, as well as the probability that the adversary is able to come up with some 'bad' ciphertext which could have potentially provided information about $m_\sigma$, is 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.

This completes a description of the games.

### A.2.2 Notation

Before we proceed to the full proof, we describe some helpful notation. Most of this notation is borrowed from [CS03]. Let $\mathcal{A}$ be the DR-CCA adversary. We set the security parameter to be $\lambda \in \mathbb{Z}_{\geq 0}$ and the group description $\Gamma[\hat{\mathbb{G}}, \mathbb{G}, g, q] \in [S_\lambda]$.

Suppose that the public key is $(\Gamma, \mathsf{hk}, g_1, g_2, c, d, h)$ 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 $x, y$ and $z$ 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 $\psi$, 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 $t = x_1 r + y_1 r\alpha + x_2 \hat{r} w + y_2 \hat{r} \alpha w$.

We will also deal with the target ciphertext $\psi^*$. When dealing with this particular ciphertext, we denote each of the variables with a $*$, to obtain $u_1^*, u_2^*, e'^*, e^*, v^*, r^*, \hat{r}^*, \alpha^*, r_e^*, r_v^*, t^*$.

### A.2.3 Proof of Theorem A.1

We now proceed to describe the games in detail. Game $\mathbf{G}_0$ is the original attack 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 $\psi^*$, we use a modified encryption algorithm, in which the steps **E4** and **E7** are replaced by

$\mathbf{E4'}: e' \leftarrow u_1^{z_1} u_2^{z_1}$;

$\mathbf{E7'}: v \leftarrow u_1^{x_1+y_1\alpha} u_2^{x_2+y_2\alpha}$.

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 is our first new game. This game is identical to $\mathbf{G}_1$ with the following modification.

We modify the *encryption* oracle in Stage 3 such that along with a ciphertext $\sigma^*$, it also outputs the *auxiliary information* $\mathsf{aux}_1 = (g_1^{r^2}, g_2^{r^2}, (h^r \cdot m_i)^r, (cd^\alpha)^{r^2}) = (u_1^{*r}, u_2^{*r}, e^{*r}, v^{*r})$. 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 the adversary the values $(g_1^{r^2}, g_2^{r^2}, (h^r \cdot m_i)^r, (cd^\alpha)^{r^2})$ *before* it can no longer make decryption oracle queries, we can only increase the power (and hence the advantage) of the adversary. It follows that $\mathsf{AdvDRCCA}_1(\lambda) \leq \mathsf{AdvDRCCA}_2(\lambda)$. Rephrasing, we can see that

$$|\Pr[T_1] - 1/2| \leq |\Pr[T_2] - 1/2|$$

and hence it is enough to show that $|\Pr[T_2] - 1/2| \leq \mathsf{negl}(\lambda)$. The assertion will follow immediately.

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

Note that the only difference between the games $\mathbf{G}_2$ and $\mathbf{G}_3$ is that the tuple $(g_1, u_1, \mathsf{aux}_1[0])$ is a uniformly distributed tuple from $\mathsf{SDH}_{\lambda,\Gamma}$ in $\mathbf{G}_2$ while it is a uniformly distributed tuple from $\mathsf{RandSDH}_{\lambda,\Gamma}$ in $\mathbf{G}_3$. It is thus immediately clear that the indistinguishability of the two games follows from the hardness of SDH. More specifically, we show the following.

**Lemma A.1.** *There exists some polynomial-time probabilistic algorithm $\mathcal{A}_{\mathsf{SDH}}$ such that*

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

*Proof.* We will describe $\mathcal{A}_{\mathsf{SDH}}$ in detail. The algorithm takes in as input $1^\lambda$, a group description $\Gamma[\hat{\mathbb{G}}, \mathbb{G}, g, q] \in [S_\lambda]$ and some tuple $(g^a, g^b)$. The algorithm interacts with the DR-CCA adversary $\mathcal{A}$. First, it begins by computing

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

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 $\mathsf{PK}$ to $\mathcal{A}$. We now show the behaviour of $\mathcal{A}_{\mathsf{SDH}}$ 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 $\mathsf{SK}$ to decrypt and outputs either $\perp$ or some $m$. Whenever $\mathcal{A}$ submits $(m_0, m_1)$ to the encryption oracle, $\mathcal{A}_{\mathsf{SDH}}$ samples a uniform $\sigma \xleftarrow{\$} \{0,1\}$ and computes

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

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 $\mathsf{halt}$ to the auxiliary oracle, $\mathcal{A}_{\mathsf{SDH}}$ outputs $(x_1, x_2, y_1, y_2)$.

Once the game in $\mathbf{G}_2$ has been perfectly simulated, the adversary outputs 1 if $\hat{\sigma} = \sigma$ and 0 otherwise. It is clear from the above simulation that for any fixed $\lambda$ and $\Gamma$,

$$\Pr[T_2] = \Pr[\mathcal{A}_{\mathsf{SDH}}(1^\lambda, \Gamma, \rho) = 1 : \rho \xleftarrow{\$} \mathsf{SDH}_{\lambda,\Gamma}]$$
$$\Pr[T_3] = \Pr[\mathcal{A}_{\mathsf{SDH}}(1^\lambda, \Gamma, \rho) = 1 : \rho \xleftarrow{\$} \mathsf{RandSDH}_{\lambda,\Gamma}].$$

It immediately follows that

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

□

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

The proof essentially follows the same as that of the previous game, and can be omitted.

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

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

In the games up to $\mathbf{G}_4$ we had $r^* = \hat{r}^*$, however in game $\mathbf{G}_5$ $r^*$ and $\hat{r}^*$ are completely independent except that they cannot be equal. Note, however, that like in the previous game, we have a tuple $(g_1, g_2, u_1^*, u_2^*)$ which in game $\mathbf{G}_4$ is uniformly sampled from $\mathsf{DH}_{\lambda,\Gamma}$ while in game $\mathbf{G}_5$ is uniformly sampled from $\mathsf{RandDH}_{\lambda,\Gamma}$. This leads to the following lemma, which says that any distinguisher for the two games immediately gives a statistical distinguisher for the DDH instance.

**Lemma A.2.** *There exists some probabilistic-polynomial time 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 adversary $\mathcal{A}_{\mathsf{DDH}}$ is initiated with $1^\lambda$, a group description $\Gamma$, and a tuple $(g_2, u_1, u_2) \in \mathbb{G}^3$.

The adversary $\mathcal{A}$ is used in the same way, with the decryption and auxiliary oracles having the same description. The only difference is how the encryption oracle responds with the target ciphertext. In this case, when $\mathcal{A}_{\mathsf{DDH}}$ receives $(m_0, m_1)$, it computes $\sigma \xleftarrow{\$} \{0, 1\}$ and the following values:

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

The auxiliary information $\mathsf{aux}_1$ is sampled in the same manner, with $g^s$ and $g^{s'}$ uniformly random elements of $\mathbb{G}$.

Once the game in $\mathbf{G}_4$ has been perfectly simulated, the adversary outputs 1 if $\hat{\sigma} = \sigma$ and 0 otherwise. It is clear from the above simulation that for any fixed $\lambda$ and $\Gamma$,

$$\Pr[T_4] = \Pr[\mathcal{A}_{\mathsf{DDH}}(1^\lambda, \Gamma, \rho) = 1 : \rho \xleftarrow{\$} \mathsf{DH}_{\lambda,\Gamma}]$$
$$\Pr[T_5] = \Pr[\mathcal{A}_{\mathsf{DDH}}(1^\lambda, \Gamma, \rho) = 1 : \rho \xleftarrow{\$} \mathsf{RandDH}_{\lambda,\Gamma}].$$

It immediately follows that

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

□

**Game $\mathbf{G}_6$.** This is game $\mathbf{G}_3$ of the proof of [CS03]. In this game, we modify the decryption oracle. Instead of using the original decryption oracle, we modify the oracle, replacing the 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′**: $h \leftarrow u_1^z$.

We first notice that the decryption oracle does not make any use of $x_i, y_i, z_i$ except indirectly. 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 refuse to answer 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 $\mathbf{G}_6$.

We recall the following, which is Lemma 4 in [CS03].

**Lemma A.3.** *Let $U_1$, $U_2$ and $F$ be events defined on some probability space. Suppose that the event $U_1 \wedge \neg F$ occurs iff $U_2 \wedge \neg F$ occurs. Then $|\Pr[U_1] - \Pr[U_2]| \leq \Pr[F]$.*

We will apply this lemma in our analysis. Note that the event $T_5 \wedge \neg R_6$ and $T_6 \wedge \neg R_6$ are identical. By the lemma, we get

$$|\Pr[T_3] - \Pr[T_2]| \leq \Pr[R_3]$$

and it suffices to bound $\Pr[R_3]$. This will be done in games $\mathbf{G}_7$ and $\mathbf{G}_8$.

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

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

It follows that $\Pr[T_7] = 1/2$, because now $\psi^*$ does not even involve $\sigma$, so the adversary's output is completely independent of it. We define the event $R_7$ which is the same as $R_6$, ie. some ciphertext $\psi$ is submitted to the decryption oracle which is rejected by **D4′** but would have passed **D4**.

**Lemma A.4.** *We have*

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

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

**Lemma A.5.** *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 $i$th rows of $M$ and $\gamma$ 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 $\beta_k$ is uniformly distributed over $K$ in the resulting probability space.*

We will require a less general version of this lemma, but it suffices for the proof below.

*Proof (of Lemma A.4).* 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 $X = (\mathsf{coins}, \mathsf{hk}, w, x_1, x_2, y_1, y_2, \sigma, r^*, \hat{r}^*)$, where $\mathsf{coins}$ is the internal randomness of the PPT adversary $\mathcal{A}$, and the quantity $z$. 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 can 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 $\mathsf{aux}$ as well. Hence, knowledge of $\mathsf{aux}$ has no bearing on the independence of $r_7^*$, and it remains so even if $\mathsf{aux}$ is public.

We show that the same is true for $r_3^*$. Consider

$$\begin{pmatrix} z \\ r_3^* \end{pmatrix} = \begin{pmatrix} 1 & w \\ r^* & w\hat{r}^* \end{pmatrix} \cdot \begin{pmatrix} z_1 \\ z_2 \end{pmatrix} + \begin{pmatrix} 0 \\ \log_g m_\sigma \end{pmatrix}.$$

Let the $2 \times 2$ matrix be referred to as $M$. 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 $\mathsf{aux}$). Furthermore, $\det(M) = w(\hat{r}^* - r^*) \neq 0$. The proposition then follows from Lemma A.5. □

**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 the adversary submits a ciphertext $\psi$ 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 $\mathsf{reject}$ and halts even before executing $\mathbf{D4}'$.

We define two events:

1. $C_8$, which is the event that the adversary submits a ciphertext which is rejected by the special rule.
2. $R_8$, which is the even that some ciphertext $\psi$ is submitted which is is passed by the special rule, however it is rejected by $\mathbf{D4}'$, and would have been accepted by $\mathbf{D4}$.

Note that the games proceed identically till $C_8$ occurs. Particularly, $R_7 \wedge \neg C_8$ and $R_8 \wedge \neg C_8$ are identical. We have, using lemma Lemma A.3 that

$$|\Pr[R_5] - \Pr[R_4]| \leq \Pr[C_5].$$

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

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

**Lemma A.7.** *We have*

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

The proof of Lemma A.6 is identical to the proof of Lemma 7 presented in [CS03], hence we will not present it. The proof of Lemma A.7 is also similar, however must be adapted to our setting.

Together, we show that $|\Pr[R_8] - \Pr[R_6]| \leq \mathsf{negl} \implies |\Pr[T_6] - 1/2| \leq \mathsf{negl}$, which proves the assertion that $|\Pr[T_0] - 1/2| \leq \mathsf{negl}$. We complete the proof with of Lemma A.7 below.

*Proof (of Lemma A.7).* Consider the proof of Lemma 8 presented in [CS03]. Both sub-parts of the proof rely on the fact that the vector $\mathsf{aux} = (x_1, x_2, y_1, y_2)$ is independent and uniformly distributed in the eyes of the adversary. Clearly this is not the case if $\mathsf{aux}$ is provided as auxiliary information. However, the proof only relies on the fact that $\mathsf{aux}$ 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, $\mathsf{aux}$ is uniform and independent, and the proof of Lemma 8 follows without any additional concerns. □

With this, we have shown that the Cramer-Shoup encryption scheme as defined in Fig. 1 is DR-CCA secure with auxiliary information $\mathsf{aux} = (x_1, x_2, y_1, y_2, g_1^{r^2}, g_2^{r^2}, (h^r \cdot m_i)^r, (cd^\alpha)^{r^2})$.