On the UC-(In)Security of PAKE Protocols in the Plain Model

Naman Kumar, Jiayu Xu

November 6, 2024

This document contains a proof that the PAKE protocol of Katz, Ostrovsky and Yung (2001) is not UC-secure.

1 UC-(In)Security of the KOY Protocol

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 crs can be interpreted to be a degenerate Cramer-Shoup public key with an unknown secret key.

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

- Upon receiving $(\mathsf{VK}, A, B, C, D)$, the Server samples its 'Diffie–Hellman exponents' (x_2, y_2, z_2, w_2), and computes $E = g_1^{x_2} g_2^{y_2} h^{z_2} (cd^\alpha)^{w_2}$.[1] Furthermore, Server encrypts pw with label (msg_1, E) using Cramer–Shoup encryption as done by User in msg_1. Let r_2 be the randomness used in encryption. The resulting ciphertext is $\mathsf{ct}_2 := (F, G, I, J) = (g_1^{r_2}, g_2^{r_2}, h^{r_2} \cdot \mathsf{pw}, (cd^\beta)^{r_2})$. Server sends $\mathsf{msg}_2 = (E, \mathsf{ct}_2)$ to User.

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

- Server verifies the signature st and aborts if it is invalid. Otherwise the session key sk is defined as the product of

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

  and

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

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

User 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, User 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.[2] 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 $\mathsf{sk} = X_1X_2$); 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.

The one-time signature scheme effectively forces the adversary to pass msg_2 and msg_3 without modification as long as it passes msg_1. This is to prevent a man-in-the-middle adversary from gaining information by passing all ciphertexts but modifying the rest of the messages. Specifically, (if the signature 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 α. In this way if the adversary changes VK while keeping the ciphertext $A|B|C|D$, the ciphertext would become invalid.

1.2 Technical Overview

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

UC-insecurity of KOY. Our attack relies on an adversary $\mathcal{A}$ that completely disregards the presence of Server and instead interacts with User while executing Server's algorithm on its own. In particular, once the protocol is initiated by User, $\mathcal{A}$ 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 User's password pw (i.e., we assume that $\mathcal{A}$ makes a correct password guess) and computes $\mathsf{msg}_2 = E|F|G|I|J$. User 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 User.

To see why a simulator $\mathcal{S}$ cannot simulate this adversary, we attempt an ideal-world execution and pinpoint where it fails. Since $\mathcal{S}$ is allowed to choose $\mathsf{crs} = (g_1, g_2, h, c, d)$, it can sample g_1 at random and set h such that $h = g_1^{\ell}$ — in other words, $\mathcal{S}$ chooses the "CRS trapdoor" $\ell = \log_{g_1} h$. After receiving the NewSession command from $\mathcal{F}_{\mathsf{PAKE}}$, $\mathcal{S}$ must simulate User's first message msg_1. Since $\mathcal{S}$ does not know the password, at this point it must (effectively) guess some pw′ at random; that is, in msg_1, $C = h^{r_1} \cdot \mathsf{pw}'$ where pw′ can be no better than a random password sampled from the dictionary. (C is indistinguishable from the correct value due to the security of Cramer–Shoup encryption.) After $\mathcal{Z}$ responds with $\mathsf{msg}_2 = E|F|G|I|J$, since $F = g_1^{r_2}$ and $I = h^{r_2} \cdot \mathsf{pw}$, $\mathcal{S}$ can extract pw as I/F^{ℓ}. Once this has been done, $\mathcal{S}$ can send a TestPwd command to $\mathcal{F}_{\mathsf{PAKE}}$ on the correct pw; $\mathcal{F}_{\mathsf{PAKE}}$ would mark the User session compromised and thus allow $\mathcal{S}$ to choose User's session key sk (which has to be consistent with the session key User computes in the real world).[3] This is where the game-based security and UC-security

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

[3] A TestPwd *must* be run, since we require that msg_1 and msg_2 together with the randomness of the User and $\mathcal{A}$ together

of PAKE diverge: in game-based security, all security guarantees are considered lost (and the simulation of the game can stop) once the adversary guesses the correct password; whereas in UC-security the simulation has to continue. The problem here is that *even knowing the correct password* pw*, $\mathcal{S}$ still cannot determine what* sk *should be.*

Recall that sk is the product of

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

and

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

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

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 msg_2. At this point all group elements that $\mathcal{Z}$ has seen are g_1, g_2, h, c, d from crs, A, B, C, D from msg_1, and pw. An algebraic $\mathcal{Z}$ must "explain" how E is computed; for now let's ignore A, B, C, D, pw and assume $\mathcal{Z}$ computes

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

In the real world User would compute $X_1 = E^{r_1}$, which is equal to $A^{x_2} B^{y_2} (C/\mathsf{pw})^{z_2} D^{w_2}$. In the ideal world, as explained above, $\mathcal{S}$ cannot compute X_1 as E^{r_1} since it chose the wrong pw' with high probability while generating A, B, C, D; however, after extracting the correct pw from msg_2, $\mathcal{S}$ can still compute X_1 as $A^{x_2} B^{y_2} (C/\mathsf{pw})^{z_2} D^{w_2}$, since now it sees the algebraic coefficients x_2, y_2, z_2, w_2 from $\mathcal{Z}$. In this way $\mathcal{S}$ generates X_1 that is indistinguishable from the real world.

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

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

In the real world User would compute

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

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

determine sk; allowing the simulation to proceed without a $\mathsf{TestPwd}$ would result in $\mathcal{F}_{\mathsf{PAKE}}$ outputting a uniformly random key.

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 User by running Server's algorithm on User's password pw. When User's session completes, the environment $\mathcal{Z}$ sees User'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 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 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[4], then $\mathcal{Z}$ can recover r_1 after User's session completes, and check if $C = h^{r_1} \cdot \mathsf{pw}$. In the real world this equation holds, whereas in the ideal world it does not if $\mathcal{S}$ chose the "wrong" $\mathsf{pw}' \neq \mathsf{pw}$ to encrypt while simulating msg_1. In fact, it seems that for the KOY protocol to be UC-AGM-secure, we need the Jiayu: FILL IN assumption. Jiayu: I think the assumption should be Decisional Square Diffie–Hellman (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* User*'s session*. Indeed, the reduction $\mathcal{R}$ to the security of Cramer–Shoup roughly works as follows:

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

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

   (c) Finally, if the password guess in msg_2^* is correct, $\mathcal{R}$ needs to simulate sk_U. (This step is not needed in the game-based proof.)[5]

   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 Jiayu: FILL IN Jiayu: I think this should work, but it needs to be double checked

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. Jiayu: I hope so...

[4] 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 [?].

[5] We omit the remaining steps which are to simulate msg_3, and on msg_3^* simulate Server's session key sk_S, as they are inconsequential to our main point.

1.3 Proof of UC-Insecurity

Theorem 1.1. *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 1 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.

<u>Environment $\mathcal{Z}$:</u>

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 $(\mathsf{NewSession}, \mathsf{sid}, \mathsf{User}, \mathsf{Server}, \mathsf{pw})$ to User.
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 User.
3. $\mathcal{Z}$ receives $\mathsf{msg}_3 = \mathsf{sid}|K|\sigma$ from $\mathcal{A}$ and $(\mathsf{sid}, \mathsf{sk})$ from User.
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 1: Our Setup.

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 User and Server combined):

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

It then sets $\mathsf{pw} := C/h^b$ and samples $x_2, y_2, w_2, r_2 \xleftarrow{\$} \mathbb{Z}_q$, setting

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

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

3. $\mathcal{R}$ receives $\mathsf{msg}_3 = \mathsf{sid}|K|\sigma$ (as the second message from User to Server) and $(\mathsf{sid}, \mathsf{sk})$ from $\mathcal{S}$ (as User'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 1; the difference is that $\mathcal{Z}$ samples pw and z_2 on its own, whereas $\mathcal{R}$ sets $\mathsf{pw} = C/h^b$ and $z_2 = a$ — which cannot be detected by $\mathcal{S}$. Let $C' = C/\mathsf{pw} = h^b$ and

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

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

2 Preliminaries

2.1 Cramer-Shoup Encryption Scheme

References

[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.

A Cramer-Shoup Security

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

We define the delayed-reveal-CCA advantage with auxiliary information aux of the adversary $\mathcal{A}$ against PKE at λ, denoted as $\mathsf{AdvDRCCA}(\lambda)$ to be $|\Pr[\sigma = \hat{\sigma}] - 1/2|$ in the above attack game with corresponding value of aux. We say that PKE is delayed-reveal-CCA secure with auxiliary information aux if there exists some negligible function negl such that for all λ,

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

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 (Naman: to define) is delayed-reveal-CCA secure with auxiliary information* $\mathsf{aux} = (x_1, x_2, y_1, y_2, g_1^{r^2}, g_2^{r^2})$.

Naman: Brief Sketch of the Argument

- Game 1 is the same.
- Game 2 sets the $(g_1^{r^2}, g_2^{r^2})$ as an output of the encryption oracle. Note that this can only increase the power of the adversary, as the adversary is also allowed to make decryption queries that involve $(g_1^{r^2}, g_2^{r^2})$. It follows that $|\Pr[T_1] - 1/2|| \leq |\Pr[T_2] - 1/2|$.
- Game 3 replaces $g_i^{r^2}$ with random values, which follows from SDH.Naman: Think about this more, do the DDH and SDH have to be 'compatible'??.
- Game 4 is Game 2.
- Game 5 is Game 3.
- Game 6 is Game 4. The argument in lemma 6 holds even if X is completely public, ie. it only requires z_1 and z_2 to be uniformly and randomly distributed (and hence 'unknown'). This is implicit in lemma 6 – we just need to point it out.
- Game 7 is Game 5. The arguments in lemmas 7 and 8 hold if certain values of X are unknown, but these values only need to be unknown at the time of the decryption oracle query, which is true. Hence, the probability that a 'bad' decryption oracle query is still negligible, and both the lemmas follow (relatively) unchanged.

This should complete the proof. Naman: Complete the whole proof.

Naman: Full proof begins here. First explain notation, then we proceed as follows.

The overall strategy of our proof is borrowed from that in Cramer and Shoup's paper [CS03]. We will borrow the same notation (Naman: explain this) and proceed through a sequence of games. Apart from certain changes, the nature of our games will be similar to those in the original paper.

We recall the following lemma from [CS03].

Lemma A.1. *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 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 ψ^*, we use a modified encryption algorithm, in which the steps **E4** and **E7** are replaced by

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

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 σ^*, it also outputs the *auxiliary information* $\mathsf{aux}_1 = (g_1^{r^2}, g_2^{r^2}) = (u_1^{*r}, u_2^{*r})$. 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 $(g_1^{r^2}, g_2^{r^2})$ *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 $(\psi^*, \mathsf{aux}_1) = ((u_1^*, u_2^*, e^*, v^*), (g_1^{r^2}, g_2^{r^2}))$, the oracle samples a uniform s from $\mathbb{Z}_q$ and outputs $(\psi^*, \mathsf{aux}_1) = ((u_1^*, u_2^*, e^*, v^*), (g_1^s, g_2^{r^2}))$.

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 SDH tuple in $\mathbf{G}_2$ while it is a uniformly distributed tuple 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 claim the following.

Lemma A.2. *There exists some polynomial-time probabilistic algorithm $\mathcal{A}_{\mathsf{SDH}}$ such that*

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

Naman: Write the SDH assumption in the preliminaries, then rephrase this part of the argument depending on how it is defined there. Complete the proof as well.

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

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

Key Generation: On input 1^λ 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^λ 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 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^λ 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 ψ and a label label, do the following.

D1: Parse ψ as a 4-tuple $(u_1, u_2, e, v) \in \mathbb{G}^4$, output reject and halt if ψ is not of this form.
D2: Test if u_1, u_2 and e belong to $\mathbb{G}$; output 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 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 2: The Cramer-Shoup Encryption Scheme with Labels.

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 PK.

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

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 σ^*.

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

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

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

Figure 3: The Delayed-Reveal-CCA game.