

KOY is not UC-Secure

Naman Kumar

March 12, 2024

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

1 Overview

At a high level, 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 henceforth) and receives $\text{msg}_1 = \text{VK}|A|B|C|D$. After this, \mathcal{A} runs the server's algorithm on the correct password pw and computes $\text{msg}_2 = E|F|G|I|J$. **User** then runs its session-key generating algorithm and outputs its session key $\text{sk} = E^{r_1} F^{x_1} G^{y_1} (I')^{z_1} J^{w_1}$. We note that at this point, \mathcal{A} (and \mathcal{Z}) have all the information they need to run the server's session-key generating algorithm and ensure that their generated session key is equal to the sk generated by **User**.

To see why a simulator \mathcal{S} cannot simulate this adversary, we attempt an ideal-world execution and pinpoint where our simulation fails. Since \mathcal{S} is allowed to choose $\text{crs} = (\mathbb{G}; g_1, g_2, h, c, d \in \mathbb{G}; H : \{0, 1\}^* \rightarrow \mathbb{Z}_q)$, it can sample g_1 at random and set h such that $h = g_1^\ell$. After receiving the **NewSession** command from $\mathcal{F}_{\text{PAKE}}$, \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 , $C = h^{r_1} \cdot \text{pw}^*$ where pw^* can be no better than a random password sampled from the dictionary. After \mathcal{Z} responds with $\text{msg}_2 = E|F|G|I|J$, \mathcal{S} can extract pw as I/F^ℓ . Once this has been done, \mathcal{S} can send a **TestPwd** command to $\mathcal{F}_{\text{PAKE}}$ which would mark the **User** session compromised and thus allow \mathcal{S} to successfully choose the sk output by **User**.¹ The problem is that even with this information, \mathcal{S} still cannot determine what sk is.

¹Recall that 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 determine sk ; allowing the simulation to proceed without a **TestPwd** would result in $\mathcal{F}_{\text{PAKE}}$ outputting a uniformly random key.

To determine sk , \mathcal{S} can either run **Server**'s algorithm or **User**'s algorithm to generate session keys. Recalling that $\text{msg}_2 = E|F|G|I|J$ is provided to \mathcal{S} directly from the environment, \mathcal{S} cannot use **Server**'s algorithm since it would require the determination of all of x_2, y_2, z_2, r_2, w_2 , which is computationally infeasible under the hardness of CDH. Alternately, \mathcal{S} can run **User**'s algorithm to determine sk , for which it already has access to x_1, y_1, z_1, r_1, w_1 and E, F, G, I, J . However, we recall that this sk output by \mathcal{S} must be equal to that which \mathcal{Z} determines using its own execution of **Server**'s algorithm (this is what happens in the real world). The problem here is that the password guess pw^* that \mathcal{S} uses while generating msg_1 is likely incorrect; in fact, we can show that the output of **Server**'s algorithm on $\text{msg}_1, \text{msg}_2$ and msg_3 by \mathcal{Z} will have $(\text{pw}^*/\text{pw})^{z_2}$ as a factor, and except in the $1/|\mathcal{D}|$ probability case that $\text{pw}^* = \text{pw}$, \mathcal{S} cannot determine z_2 assuming the hardness of CDH and thus will be unable to determine sk .

2 Proof

Theorem 2.1. *Assuming the hardness of fixed-CDH, the protocol of [KOY] does not UC-realize $\mathcal{F}_{\text{pake}}$ in the \mathcal{F}_{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.

Assume that there exists a negligible function $\varepsilon := \varepsilon(\lambda)$ such that there exists a simulator \mathcal{S} for which \mathcal{Z} outputs 1 with probability $1 - \varepsilon$ in the ideal world. Before the formal execution of the protocol begins, \mathcal{S} samples $\text{crs} = (\mathbb{G}; g_1, g_2, h, c, d \in \mathbb{G}; H : \{0, 1\}^* \rightarrow \mathbb{Z}_q)$. First, assume that CDH is hard over (\mathbb{G}, p, h) . Given any two elements (h^a, h^b) where $a, b \xleftarrow{\$} \mathbb{Z}_p$, consider any reduction \mathcal{R} which does the following.

1. \mathcal{R} waits to receive $(\text{NewSession}, \text{sid}, \text{User}, \text{Server}, \text{pw})$ from **User**.
- 2.

//everything below is deprecated comments.

First, we will assume the hardness of CDH over the group (\mathbb{G}, g, p) . Let g^a, g^b be two elements where $a, b \xleftarrow{\$} \mathbb{Z}_p$.

Formally, assume that there exists a simulator \mathcal{S} such that \mathcal{Z} always outputs 1 in the ideal world. *Jiayu: Formally we cannot really assume this; need to say “such that \mathcal{Z} outputs 1 with all but negligible probability in the ideal world”. I am not entirely sure for now, but we probably need to be more specific and say “there is a negligible function ϵ such that \mathcal{Z} outputs 1 with probability $1 - \epsilon$ in the ideal world.”* We will use this simulator

Environment \mathcal{Z} :

1. \mathcal{Z} selects $\text{pw} \xleftarrow{\$} \mathcal{PW}$, where $\mathcal{PW} \subseteq \mathbb{G}$ is the password dictionary. It then sends $(\text{NewSession}, \text{sid}, \text{User}, \text{Server}, \text{pw})$ to User.
2. \mathcal{Z} receives $\text{msg}_1 = \text{sid}|\text{VK}|A|B|C|D$ from \mathcal{A} and samples $x_2, y_2, z_2, w_2, r_2 \xleftarrow{\$} \mathbb{Z}_q$. It then sets

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

and instructs \mathcal{A} to send $\text{msg}_2 = \text{sid}|E|F|G|I|J$ to User.

3. \mathcal{Z} receives $\text{msg}_3 = \text{sid}|K|\text{Sig}$ from \mathcal{A} and (sid, sk) from User.
4. \mathcal{Z} sets $C' = C/\text{pw}$ and then checks if $\text{Vrfy}_{\text{VK}}(\text{msg}_1|\text{msg}_2|K, \text{Sig}) = 1$. If yes, it computes $\text{sk}_S = A^{x_2} B^{y_2} (C')^{z_2} D^{w_2} K^{r_2}$ and outputs 1 if $\text{sk}_S = \text{sk}$. If either of the two checks fails, it outputs 0.

Figure 1: Our Setup.

to compute g^{ab} . Jiayu: We will construct a reduction \mathcal{R} that uses \mathcal{S} to solve the CDH problem in (\mathbb{G}, g, p) . (I think it's better to explicitly mention a reduction.) Our technique works as follows: first, we send Jiayu: everywhere you say “we do something”, change it to “ \mathcal{R} does something” $(\text{NewSession}, \text{sid}, \text{User}, \text{Server}, \text{pw})$ to $\mathcal{F}_{\text{PAKE}}$ Jiayu: conceptually I think \mathcal{R} should play the role of $\mathcal{F}_{\text{PAKE}}$ (if you are not sure what I am talking about, chat with me in our meeting or on Slack) and receive $\text{msg}_1 = \text{sid}|\text{VK}|A|B|C|D$ in response

from \mathcal{S} . We set $\text{pw} = C/g^b$ and sample $x_2, y_2, z_2, w_2, r_2 \xleftarrow{\$} \mathbb{Z}_q$.

$$\alpha' := H(PIDs|A|B|C|D)$$

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

$$F := g_1^{r_2}$$

$$G := g_2^{r_2}$$

$$I := h^{r_2} \cdot \text{pw}$$

$$\beta := H(\text{msg}_1|Server|E|F|G|I)$$

$$J := (cd^\beta)^{w_2}$$

i.e., the same computation as that of the honest server with a special choice of pw . Forwarding this to $\mathcal{F}_{\text{PAKE}}$, we receive $\text{msg}_3 = K|\text{Sig}$ and (sid, sk) in return.

□