Hybrid Protocol For Password-based Key Exchange in Three-party Setting

TingMao Chang, Jin Zhou, YaJuan Zhang, YueFei Zhu

Abstract—Modular design is a common approach for dealing with complex tasks in modern cryptology. The critical of this approach is that designing a secure hybrid protocol. In this paper, we study password-based key exchange in the three-party setting within the UC framework and design a hybrid protocol that UC-securely realizes such task. That is, we firstly define an appropriate ideal functionality $F_{3\text{-}pwKE}$ for password-based three-party key exchange. Next we partition the task into two sub-tasks, three-party key distribution and password-based two-party key exchange, and propose relevant two ideal functionalities, $F_{3\text{-}KD}$, F_{pwKE}. Finally, we present a ($F_{3\text{-}KD}$, F_{pwKE}) -hybrid protocol for password-based three-party key exchange that is proved to be UC-secure with respect to non- adaptive party corruption.

Keywords—Password-based, Universally composable, Key exchange, Three-party setting, Hybrid protocol

I. INTRODUCTION

Protocols for password-based two-party key exchange allow two parties to use their shared password in order to exchange a common session key. They are designed to be secure even when the shared secret key between two parties is a human-memorable password. Passwords are mostly used because they are easier to remember by humans than secrets keys with high entropy. However, when a party wants to communicate with many other parties, the number of password that a party needs to remember may be linear in the number of possible partners. In order to limit the number of passwords that each party needs to remember, password-based key exchange in the three-party setting, where each party only shares a password with a trusted server, have been received much attention in recent years. The main advantage of this solution is that it provides each party with the capability of communicating securely with other parties in the system while only requiring it to remember a single password. Its main drawback is that the server is needed during the establishment of all communication as in the Needham and Schroeder protocol [1].

Password-based three-party key exchange has been extensively studied in the last few years. The first work in this area was the protocol of Needham and Schroeder [1], which inspired the Kerberos distributed system [2]. Later, Bellare and Rogaway introduced a formal security model in this scenario along with a construction of the first provably-secure symmetric-key-based key distribution scheme [3]. After then, a special but important case in which the secret keys are drawn from a small set of values was considered by Michel Abdalla Pierre-Alain Fouque and David Pointcheval in [4]. In addition, they proposed a first provable-secure protocol in the three-party setting.

.A general framework for representing cryptographic protocols and analyzing their security is proposed by R.Canetti [5, 6]. The definitions of security, which is called UC security, in the UC framework, follow an approach which is referred to as “security by emulation of an ideal process”. Informally speaking, this approach proceeds as follows: firstly defining an ideal functionality which captures the basic security requirement for a task and a protocol is said to UC securely realize this task if it “emulates” the ideal

protocol for this ideal functionality. In contrast, the definition of some conventional security follows a different definitional approach which is called "security by indistinguishability", such as the definitions of AKE security and CCA security. Researches on the relationship between the indistinguishability-based definition of security and the emulation-based definition of security have become one of the significant topics in cryptography [7]. It's seen that the emulation-based definition of security is more convenient in the design of hybrid protocol. Any protocol that is proven to be UC secure is guaranteed by UC composition theorem to remain secure when run concurrently with arbitrary other protocols.

The UC composition theorem is a powerful tool for modular design and analysis of complex protocol. That is, given a complex task, firstly design an ideal functionality captures the basic idea and security requirement for such task. Then, partition the task into several, simple sub-tasks and design relevant sub-ideal functionalities for these sub-tasks. Next, designs sub-protocols that UC securely realize these sub-ideal functionalities and in addition, design a hybrid protocol that UC securely realizes the given ideal functionality assuming that evaluation of the sub-ideal functionalities is possible. Finally, use the composition theorem to argue that the protocol composed from the already-designed sub-protocols securely realizes the ideal functionality, so the given task.

In this paper, we study and formulate an appropriate ideal functionality, which captures the basic idea and security requirements of password-based three-party key exchange. Next, we partition the task for three-party key exchange into two sub-tasks. One is password-based two-party key exchange; the other is three-party key distribution. Finally, we propose a hybrid protocol that UC securely realizes such ideal functionality with respect to non-adaptive party corruption.

II. PRELIMINARIES

We propose the description of the distributed system in section *A*, sketch dictionary attacks in section *B*, and recall some writing conventions for ideal functionality and the definition of hybrid protocol within the UC framework in section *C*.

A. Participants and initialization

For simplicity, we model the distributed system as a fixed polynomial-size set of m client parties $\Pi = \{P_1,...,P_m\}$. The number m may be any polynomial function of the security parameter k. In addition, there is a trusted server P_T which is not a member of Π. Each party P_i has a long-term password sk_i, while P_T holds a vector $sk_T=(sk_i)$. Any two parties of Π together with P_T are allowed to run the three-party key exchange protocol at any time (possibly concurrently) in order to obtain a session key.

B. Dictionary attacks

Password-based key exchange protocols assume a more realistic scenario in which secret keys are not uniformly distributed over a large space, but rather chosen from a small set of possible values (i.e., dictionary). One problem is that they are often subject to so-called dictionary attack. Dictionary attacks are attacks in which an adversary tries to break the security of a scheme by a brute-force method, and tries all possible combinations of secret keys in a given small set of possible values (i.e., dictionary). Such attacks are usually divided in two categories: off-line and on-line dictionary attacks.

To address this problem, several protocols have been designed to be secure even when the secret key is a password. In this setting, an attacker has a noticeable chance of impersonating one of the parties simply by guessing the correct password and running the prescribed protocol. Such an attack is called an on-line dictionary attack, since one of the parties must be on-line and ready to participate in the protocol as the attacker exhaustively enumerates the dictionary in this way. Then, guessing the password in these protocols means that adversary must modify and replace the transmitted messages. Therefore, failed guess can be easily detected by participating parties.

C. Some useful notions in the UC framework

Delayed output. We often want to capture the fact that outputs generated by interactive protocol may be delayed due to delays in message delivery. We say that an ideal functionality F sends a delayed output v to some party P if it engages in the following interaction: Instead of simply outputting v to P, F first sends to the adversary a note that it is ready to generate an output to P. If the output is **public**, then the value v is included in the note to the adversary. If the output is **private,** then v is not mentioned in this note. Furthermore, the note contains a unique identifier that distinguishes it from all other messages sent by F to the adversary in this execution. When the adversary replies to the note, F outputs the value v to P.

Party corruptions. Adaptive party corruptions, namely corruptions that occur as the computation proceeds, based on the information gathered by the adversary so far. Arguably, adaptive corruption of parties is a realistic threat in existing networks. Nonetheless, it is sometimes useful to consider also a weaker threat model, where the identities of the adversarially controlled parties are fixed before the computation starts; this is the case of non-adaptive (or, static) adversaries.

Hybrid protocol. Hybrid protocol is a special type of protocol in the UC framework. In addition to communicating via the adversary in the usual way, the parties also make calls to the instances of ideal functionalities. This is done in a straightforward way, by calling the corresponding instance of the ideal protocol for these ideal functionalities.

III. DEFINITION OF IDEAL FUNCTIONALITY

In this section, we formulate UC definition of security for password-based three-party key exchange.

Given a domain G of key space, the basic idea of three-party key exchange is to allow two uncorrupted client parties to exchange the same session key that is chosen from the domain G by the help of a trusted server, as long as the shared password between each client party and trusted server is identical. However, if any one of parties is corrupted or the adversary correctly guesses any one of two passwords, then the adversary is given the power to fully determine the session key. It's natural that if corruption occurs, the adversary is unnecessary to guess password.

Ideal functionality $F_{3\text{-}pwKE}$

$F_{3\text{-}pwKE}$ proceeds as follows, given a domain G of key space:

Trustedserver: Upon receiving a query (*Trustedserver sid*,P_T) from any party P_T, record (*Trustedserver*,*sid*,P_T) and send this record to all the parties and adversary S.

Initialization: Upon receiving a query (*NewSession*,*sid*, *pid*,P_i,pw_i,*role*) from any party P_i ($P_i \neq P_T$), if *pid*=(P_i, P_j,P_T) for P_T and some party P_j, then record (*sid*,*pid*, P_i,pw_i) and send (*sid*,*pid*,P_i,*role*) to S. Else, ignore this query. Upon receiving a query *(NewSession,sid,pid,*P_T, $pw_i^{'}$, $pw_j^{'}$ *role)* from party P_T, if *pid*=(P_i,P_j,P_T), then record *(sid,pid,*P_T $pw_i^{'}$, $pw_j^{'}$) and send (*sid*,*pid*,P_T,*role*) to S. If there are already three tuples of the form (*sid*,*pid*,P_i,pw_i), (*sid*,*pid*,P_j,pw_j) and *(sid,pid,*P_T, $pw_i^{'}$, $pw_j^{'}$), record (*Session*,*sid*,*pid*,*ready*) and send it to S.

TestPassword: Upon receiving a query *(TestPwd, sid,pid,*P_i, *pw')* from the adversary S: If there is a record of the form *(sid,pid,*P_i,pw_i*)* which is **fresh**, then do:

1. If pw_i=*pw'*, mark the record **compromised** and reply to S with "**correct guess**".
2. If $pw_i \neq pw'$, mark the record **interrupted** reply to S with "**wrong guess**".

Keygeneration: If the triple (*Session*,*sid*,*pid*,*ready*) has been recorded, $F_{3\text{-}pwKE}$ choose $\kappa \xleftarrow{r} G$, and record (*Key*,*sid*, κ).

Keydelivery: 1. Upon receiving a query *(NewKey, sid,pid,*P_i, κ'*)* from S, assume that there are records of the form (*sid*,*pid*,P_i,pw_i), (*sid*,*pid*,P_j,pw_j) and *(sid,pid,*P_T, $pw_i^{'}$, $pw_j^{'}$*)*

- If both $pw_i^{'} = pw_i$ and $pw_j^{'} = pw_j$ hold, then do:
  1) If no party is corrupted, and both (*sid*,*pid*,P_i,pw_i) and (*sid*,*pid*,P_j,pw_j) are **fresh**, send a private delayed output (*Key*,*sid*,*pid*, κ) to P_i.
  2) If no party is corrupted, and either (*sid*,*pid*,P_i, pw_i) or (*sid*,*pid*,P_j,pw_j) is **interrupted,** send a public delayed output (*Key*,*sid*,*pid*,P_i,*error*) to P_i

3) If either (sid,pid,P_i,pw_i) or (sid,pid,P_j, pw_j) is **compromised,** or any party of P_i, P_j and P_T is corrupted, send *(Key,sid,pid,* κ'*)* to P_i.

- if either $pw_i^{'} = pw_i$ or $pw_j^{'} = pw_j$ do not hold, then do:

1) If P_i is corrupted, send *(Key,sid,pid,* κ'*)* to P_i.
2) Else, send a public delayed output (*Key*,*sid*,*pid*, P_i,*error*) to P_i.

2. Upon receiving a query (*Key*,*sid*,*pid*,P_i,*error*) from the adversary S before any key has been already sent to P_i, then send (*Key*,*sid*,*pid*,P_i,*error*) to P_i.

Fig 1: The three-party key exchange ideal functionality F_{3pwKE}

We now briefly explain our ideal functionality $F_{3\text{-}pwKE}$. The full description is given in Figure 1. An instance of $F_{3\text{-}pwKE}$ deals with the generation of a single session key. The generation of multi-session key is obtained by using $\hat{F}_{3-pwKE}$, the multiple session extension of $F_{3\text{-}pwKE}$. Security for the multi-session case is guaranteed via the UC and JUC theorems [5, 6].

In the definition of $F_{3\text{-}pwKE}$, the password is chosen by the environment who then hands it to the parties as input. Since we quantify over all (polynomial-time) environments, this implies that security is preserved for all efficient password distributions, as well as when arbitrarily-related passwords are used in different sessions.

As expected, $F_{3\text{-}pwKE}$ begins with a trusted server phase. When $F_{3\text{-}pwKE}$ is notified who is a trusted server, it sends this message to all the parties as well as the adversary. Next, $F_{3\text{-}pwKE}$ waits **Newsession** message that would be sent by two client parties and a trusted server. We remark that the "role" variable in the **Newsession** message, such as (*NewSession*,*sid*,*pid*,P_i,pw_i,*role*), is included in order to let a party know its role—initiator, responder or server—in the execution. This has no effect on the security, but is needed for correct executions. Once $F_{3\text{-}pwKE}$ receives notifications from the three parties —with identical values of **sid** and **pid**, it enters a "ready" state and sends ready message to the adversary.

In the $F_{3\text{-}pwKE}$, a session is marked as **compromised** if the adversary makes a successful password guess, and the adversary can determine the session key for compromised sessions. If the adversary makes an incorrect password guess in a given session, then the session is marked as **interrupted** and $F_{3\text{-}pwKE}$ will send error message to all the uncorrupted parties in the **Keydelivery** phase. Note that in the real world, making a password guess means that adversary modifies the transferred message. Therefore, incorrect password guess can be easily detected by parties and result in error in the execution of the protocol.

Then, $F_{3\text{-}pwKE}$ chooses a session key κ uniformly at random from the domain G. Finally, this session key is delivered to the two client parties after being requested by the adversary. Note that for each client party, the adversary may send two types of **Keydelivery** request. One type is a **key message** (*Key*,*sid*,*pid*,P_i, κ'), the other is an **error message** (*Key*,*sid*,*pid*,P_i,*error*).

Two cases should be emphasized. One is that if the shared key between any client party and trusted server is not identical, $F_{3\text{-}pwKE}$ will send error message to all uncorrupted parties. The other is that $F_{3\text{-}pwKE}$ will send *(Key,sid,pid,*P_i*,error)* to P_i whenever it receives *(Key,sid, pid,*P_i*,error)* from the adversary. This fact reflects that in the execution of real-life protocol, the adversary can easily cause error because it can modify and deliver the transferred messages at will.

IV. BUILDING BLOCK

In this section we recall the definition for the cryptographic primitives that we use in the construction of hybrid protocol.

A. Decisional Diffie-Hellman assumption: DDH

Assume G is an abelian group and g is a generator. The decisional Diffie-Hellman assumption, *DDH*, states, roughly, that the distributions (g^u, g^v, g^{uv}) and (g^u, g^v, g^w) are computationally indistinguishable when u,v,w are drawn at random from G.

B. Message Authentication Code (MAC)

A message authentication code *MAC = (Tag,Ver)* is defined by the following two algorithms: (1) A generation algorithm ***Tag***, possibly probabilistic, which given a message m and a secret key *sk*, produces a tag μ; and (2) A verification algorithm ***Ver***, which given a tag μ, a message *m*, and a secret key *sk*, outputs 1 if μ is a valid tag for m under *sk* and 0, otherwise. The security notion that we need for the *MAC* scheme is strong existential unforgeability under chosen-message attacks (EU-CMA), which is based on existential unforgeability notion in [8]. In this notion, the adversary should be unable to create a new valid message-tag pair, even after seeing many such valid pairs.

C. Password-based key exchange (pwKE)

A (two-party) password-based key exchange is a protocol where two parties use their shared password in order to exchange a common session key. An ideal functionality, F_{pwKE}, for two-party password-based key exchange and a real-life protocol that UC-securely realize F_{pwKE} is proposed by R.Canetti in [9]. We restate F_{pwKE} in the appendix *A*.

D. Three-party key distribution (KD)

The main idea of three-party key distribution is to allow two client parties to obtain the same session key from trusted server, as long as the shared private key between each client party and trusted server is identical. We propose an appropriate ideal functionality, $F_{3\text{-}KD}$, for three-party key distribution and a UC-secure real-life protocol for $F_{3\text{-}KD}$ in [10]. A full description of $F_{3\text{-}KD}$ is reviewed in the appendix *B*.

V. UC-SECURE PASSWORD-BASED THREE-PARTY KEY EXCHANGE PROTOCOL

In this section, we propose a F_{pwKE},$F_{3\text{-}KD}$-hybrid protocol that UC securely realizes ideal functionality $F_{3\text{-}pwKE}$.

First, we present our generic construction of the protocol. This generic construction can be seen as a form of complier, which transfers any UC-secure password-based two-party exchange protocol into UC-secure password-based three-party key exchange protocol.

Assume *DDH* assumption is hold in group *G* and *g* is a generator, the generic construction of our protocol is described in Figure 2. The full description of our proposed protocol is described in Figure 3.

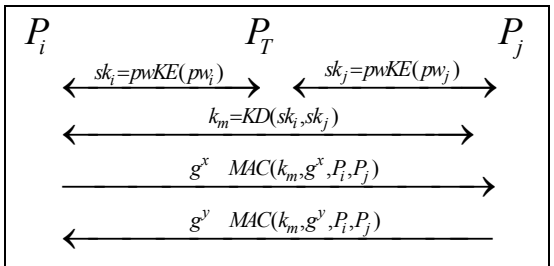

Fig 2: The generic construction of protocol

Protocol π

Assume DDH assumption is hold in group *G* and *g* is a generator, π proceeds as follows,

1. Upon party P_T is activated with a query (*Trustedserver*,*sid*,P_T), P_T send (*Trustedserver*,*sid*,P_T) to all the parties and $F_{3\text{-}KD}$. Upon P_T receiving a query *(NewSession,sid,*$ssid_1$*,*$ssid_2$*,*$ssid_3$*,*pw_i*,*pw_j*,role)*, if *role= server* and *pid*=(P_i,P_j,P_T), P_T send *(NewSession,sid,*$ssid_1$*,* pid_1*,*P_T*,*pw_i*,role) with* pid_1*=(*P_i*,*P_T*) and (NewSession, sid,*$ssid_1$*,*pid_2*,*P_T*,*pw_{ji}*,role) with* pid_2*=(*P_j *,*P_T*) to* F_{pwKE}.
2. Upon party P_i is activated with a query *(NewSession,sid,*$ssid_1$*,*$ssid_3$*,*pid_1*,*pw_i*,initiator)* , P_i send *(NewSession,sid,*$ssid_1$*,*pid_1*,*pw_i*,role) with* pid_1*=(*P_i*,*P_T*) to* $\widehat{F}_{pwKE}$. Upon party P_j is activated with a query *(NewSession,sid,*$ssid_2$*,*$ssid_3$*,*pid_2*,*pw_j*,responder)*, P_j send *(NewSession,sid,*$ssid_2$*,*pid_2*,*pw_j*,role) with* pid_2*= (*P_j*,*P_T*) to* $\widehat{F}_{pwKE}$.
3. Upon parties P_i , P_T both receiving (*sid*,$ssid_1$,pid_1 sk_i) from $\widehat{F}_{pwKE}$ and parties P_j, P_T also both receiving (*sid*,$ssid_1$,pid_1,sk_j) from $\widehat{F}_{pwKE}$, then do: P_i send (*NewSession,sid,*$ssid_3$*,pid,*P_i*,*sk_i*,role*), P_j send (*New -Session,sid,*$ssid_3$*,pid,*P_j*,*sk_j*,role*) and P_T send (*New-Session,sid,*$ssid_3$*,pid,*P_T*,*sk_i*,*sk_j*,role*) to $F_{3\text{-}KD}$. In addition, P_T sends (*Sessionkey*,*sid*,$ssid_3$,,*pid*) to $F_{3\text{-}KD}$.
4. If P_i (P_j) receives **error** message from $F_{3\text{-}KD}$, it outputs error message to all other parties and terminates. In this case, P_j also outputs error message and terminates.

5. Upon party P_i receiving (*Key,sid,ssid*$_3$*,pid,* κ_m) from $F_{3\text{-}KD}$, P_i choose x at random, computes $\alpha = g^x$, $\sigma_i = MAC(\kappa', g^x, P_i, P_j)$, and send (**flow-one,** α , σ_i) to P_j.

6. Upon party P_j receiving *(Key,sid,ssid*$_3$*,pid,* κ_m *)* from $F_{3\text{-}KD}$,, P_j choose y at random, computes $\beta = g^y$, $\sigma_j = MAC(\kappa', g^y, P_i, P_j)$. Upon receiving **flow- one** message from P_i, P_j send (**flow-two**, β , σ_j) to P_i.

7. Upon receiving **flow-two** message from P_j, P_i checks if $VF(\kappa_m, \beta, P_i, P_j, \sigma_j) = 1$. If yes, P_i compute $\kappa = \beta^x$ and output (*Key,sid,pid,* κ), else output (*Key, sid,pid,error*),. Either way P_i terminates. Upon receiving **flow-one** message from P_i, P_j checks if $VF(\kappa_m, \alpha, P_i, P_j, \sigma_i) = 1$. If yes, P_j compute $\kappa = \alpha^y$ and output *(Key,sid,pid* κ *)*, else output *(Key,sid,pid, error)*. Either way P_j terminates.

Fig 3: The three-party key exchange protocol π

VI. Proof of UC-Security

In this section, we will prove UC security of protocol π .

Theorem 1 Assume that *DDH* assumption is hold in the group *G* and (*MAC,VF*) is an EU-CMA secure message authentication code, then password-based three-party key exchange protocol π UC-securely realizes $F_{3\text{-}pwKE}$ with respect to non-adaptive party corruption. .

Proof In order to prove this theorem, we need to show that for any PPT real-world adversary *A*, there is an ideal-process adversary (simulator) *S*, such that no poly-time environment *Z* can distinguish with non-negligible probability whether it interacts with *A* and parties running π in the real world, or with *S* and (dummy) parties communicating with ideal functionality in the ideal process. The description of simulator *S* is as follows:

S will invoke an instance of adversary *A* and simulates the execution of protocol π for *A*. Thereafter, *S* makes use of the information that is sent in the simulating protocol to achieve the goal that *Z* can't distinguish ideal process and real world. Since π is ($\widehat{F}_{pwKE}$,$F_{3\text{-}KD}$*)*-hybrid protocol, except for simulating the behavior of all honest parties, *S* must simulates the behavior of those two ideal functionalities.

The simulator *S* chooses pw_i, pw_j, *sk', sk",* $\kappa_m^{'}$, while two passwords are pw_i and pw_j, two sub-session keys which $\widehat{F}_{pwKE}$ should output are *sk'* and *sk",* and sub-session key which $F_{3\text{-}KD}$ should output is $\kappa_m^{'}$. The detail of the description of *S* is as follows:

1. *S* generates pw_i, pw_j on behalf of all the parties for protocol π . Note that if either P_i or P_j is uncorrupted, *S* must know the actual password of that party. In addition, *S* chooses *sk', sk"* on behalf of $\widehat{F}_{pwKE}$ and $\kappa_m^{'}$ on behalf of $F_{3\text{-}KD}$.
2. *S* invokes an instance of *A*, then messages from *Z* to S are forwarded to *A*, and messages from *A* to *S* are forwarded to *Z*. If party P_i or P_j is corrupted, *S* sends pw_i to *A*. If $P = P_T$, *S* sends the internal state of simulated party *P* together with pw_i, pw_j to *A*.

3. Upon receiving a message (*Trustedserver*,*sid*,P_T) from $F_{3\text{-}pwKE}$, *S* sends this message to *A*. Upon receiving a message (*sid*,*pid*,P_i,*role*) from $F_{3\text{-}pwKE}$, *S* begins simulating for *A* a copy of protocol π (called $\tilde{\pi}$) with session ID is *sid* and partner ID is *pid*. In addition, *S* starts to simulate party P_i.

4. Upon receiving a message (*Session*,*sid*,*pid*,*ready*) from $F_{3\text{-}pwKE}$ (it means that the simulated protocol $\tilde{\pi}$ also is ready), *S* sends (*sid,ssid*$_1$*,pid*$_1$*,*P_i*,role*), (*sid,ssid*$_1$*,pid*$_1$*,*P_T*,role*) with $pid_1 = (P_i, P_T)$ and (*sid, ssid*$_2$*,pid*$_2$*,*P_j*,role*), (*sid,ssid*$_2$*,pid*$_2$*,*P_T*,role*) with $pid_2 = (P_j, P_T)$ to *A* on behalf of $\widehat{F}_{pwKE}$.

5. If receiving (*TestPwd,sid,ssid*$_1$*,pid*$_1$*,*P_i*,*pw_1) for P_i from the adversary *A* (it means both P_i and P_T are uncorrupted), *S* sends this message to $F_{3\text{-}pwKE}$. After then, if *S* receive "correct guess" from $F_{3\text{-}pwKE}$, it resets password of P_i to *pw',* in addition, it responds "correct guess" to *A* on behalf of $\widehat{F}_{pwKE}$ and marks (*sid,ssid*$_1$*,pid*$_1$*,*P_i*,*pw_i) as **compromised** on behalf of $\widehat{F}_{pwKE}$. Else, S respond "wrong guess" to *A* and

marks (*sid,ssid1,pid1,Pi pwi*) as **interrupted**. If receiving (*TestPwd,sid,ssid1,pid1,Pj,pw2*) for P_j from the adversary A , S does similarly as above.

6. Upon the simulated $\widehat{F}_{pwKE}$ (in fact, S) receiving (*NewKey,sid,ssid1,pid1,Pi,sk1*) from A, if P_i is corrupted or the record *(sid,ssid1,pid1,Pi,pwi)* is marked as **compromised,** S replace sk_i with sk_1. If the record (*sid,ssid1,pid1,Pi,pwi*) is marked as **interrupted**, S sends (*Key*,*sid*,*pid*,P_i,*error*) to $F_{3\text{-}pwKE}$. (Note that $\widehat{F}_{pwKE}$ must send two different sub-session key to P_i and P_T in this case, then $F_{3\text{-}KD}$ will send error message to P_i because the shared private key between P_i and P_T are not identical.) Upon the simulated $\widehat{F}_{pwKE}$ (in fact, S) receiving (*NewKey,sid,ssid2,pid2,Pj,sk2*) from A, S does similarly.

7. If S has already sent two error messages to $F_{3\text{-}pwKE}$ for P_i and P_j in step 6, it terminates the execution of $\tilde{\pi}$. Else, S sends (*sid,ssid3,pid,Pi,role*), (*sid,ssid3*, *pid*,P_j,*role*), (*sid,ssid3,pid*,P_T, *role*) and (*Session*,*sid*, *ssid3,pid*,*ready*) to A on behalf of $F_{3\text{-}KD}$.

8. Upon the simulated $F_{3\text{-}KD}$ receiving (*Key,sid,ssid3,* *pid,Pi,* κ_1) from A, If P_i is corrupted or the record (*sid,ssid1,pid1,Pi,pwi*) is marked as **compromised,** S replace $\kappa'$with κ_1. If the simulated $F_{3\text{-}KD}$ receive (*Key*,*sid*,ssid$_3$,*pid*,P_i,*error*) from A,, S sends (*Key*,*sid*, *pid*,P_i,*error*) to $F_{3\text{-}pwKE}$. S does similarly with the simulated P_j.(Note that the case that (*sid,ssid1,pid1,* P_i,*pwi*) is marked as **interrupted** is impossible in this step.)

9. If P_i is uncorrupted, S randomly chooses *x*', computes α', σ_i' and sends (flow-one, α', σ_i') to A. Else, S does nothing except waiting **flow-one** message from A. Upon the simulated party P_j receiving (flow-one, α_1, σ_1) from A, S checks if P_j is corrupted. If not, S randomly chooses *y*', computes β', σ_j' and sends (flow-two, β', σ_j') to A. Else, S does nothing except waiting **flow-two** message from A.

10. After the simulated party P_i receives a message (flow-two, α_2, σ_2) from A, then S checks if P_i is corrupted. If yes, S does nothing except waiting key message or error message for P_i from A and sends this message to $F_{3\text{-}pwKE}$. Else, if P_i accepts and outputs a session key κ', S sends (*Key,sid,pid,Pi,* $\alpha_1^{x'}$) to $F_{3\text{-}pwKE}$. If P_i outputs error message (*Key*,*sid*,*pid*,P_i,*error*), S sends (*Key*,*sid*,*pid*,P_i,*error*) to $F_{3\text{-}pwKE}$. Correspondingly, S does similarly for the simulated party P_j.

From above construction, it's easy to see that if either P_T, P_i or P_j is corrupted or A guess either pw_i or pw_j successfully, P_i (P_j) will output as A wishes both in the real world and ideal process.

Suppose that no party is corrupted. If A guess passwords wrongly, P_i and P_j will output error message both in the ideal process and in the real world. If there is no password guessing query, we argue that in the real world, Z will get no useful information of session key from party P_i or P_j and the key output by P_i or P_j is identical. The second statement hold is attributable to the DDH assumption and the EU-CMA security of *MAC*. Furthermore, P_i will output error message in the real world iff P_i will output error message in the ideal world, while P_i will output g^{xy} in the real world iff P_i will output κ in the ideal process. Since both g^{xy} and κ are chosen at random from group G, so suffice the indistinguishability. (Note that we take the non-adaptive party corruption into consideration, the case that occurs in the proof of insecurity of "class" two-move Diffie-Hellman protocol [11] is avoided.)

REFERENCES

[1] R. M. Needham and M. D. Schroeder. Using encryption for authentication in large networks of computers. Communications of the Association for Computing Machinery, 21(21):993 – 999, Dec. 1978.

[2] J. G. Steiner, B. C. Neuman, and J. L. Schiller. Kerberos: An authentication service for open networks. In Proceedings of the USENIX Winter Conference, pages 191 – 202, Dallas, TX, USA, 1988.

[3] M. Bellare and P. Rogaway: "Provably Secure Session Key Distribution –the Three Party Case" in 28th Annual ACM Symposium on Theory of Computing, pp 57–66, Philadephia, Pennsylvania, USA, May 22–24, 1996. ACM Press.

[4] M.Abdalla, P.Fouque and D.Pointcheval: "Password-Based Authenticated Key Exchange in the Three-Party Setting" in IEEE

Proceedings -- Information Security, Volume 153, issue 1, pp. 27-39, March 2006.

[5] R. Canetti: "Universally Composable Security: A New Paradigm for Cryptographic Protocols" in 42nd IEEE Symposium on Foundations of Computer Science (FOCS), IEEE, pp. 136–145, 2001.

[6] R. Canetti: "Universally Composable Security: A New Paradigm for Cryptographic Protocols 2005" in. Revision 3 of ECCC Report TR01-016

[7] R. Canetti and T. Rabin: "Universal Composition with Joint State" in Advances in Cryptology Crypto 2003, LNCS vol. 2729, Springer–Verlag, pp. 265–281, 2003.

[8] M. Bellare, J. Kilian, and P. Rogaway. The security of the cipher block chaining message authentication code. Journal of Computer and System Sciences, 61(3):362 – 399, 2000. (Cited on page 14.)

[9] J. Katz, J. Sun Shin. "Modeling Insider Attacks on Group Key-Exchange Protocols" in http://eprint.iacr.org/2005/163

[10] Jin Zhou, TingMao Chang, YaJuan Zhang, YueFei Zhou. Universally Composable Three-Party Key Distribution. in http://eprint.iacr.org/2006/421

[11] R. Canetti and H. Krawczyk: "Universally Composable Notions of Key Exchange and Secure Channels" in Eurocrypt 2002.Full version available at http://eprint.iacr.org/2002/059.

APPENDIX

A Ideal functionality F_{pwKE}

Ideal functionality F_{pwKE}.

Initialization: Upon receiving a query *(NewSession,sid, pid,P_i,pw_i,role)* with *pid*=(P_i,P_j) from party P_i , send (*sid,pid,* P_i,role) to *S*. In addition, if this is the first New Session query, or if this is the second New Session query and there is a record *(sid,pid,P_j pw_j)*, then record *(sid,pid,P_i pw_i)* and mark this record **fresh.**

TestPassword: Upon receiving a query *(TestPwd,sid, pid,P_i;pw')* from the adversary *S*: If there is a record of the form *(sid,pid,P_i pw_i)* which is **fresh**, then do:

1. If pw_i = *pw'*, mark the record **compromised** and reply to *S* with "**correct guess**".
2. If $pw_i \neq pw'$, mark the record **interrupted** and reply with "**wrong guess**".

Keydelivery: Upon receiving a query *(NewKey,sid,pid,P_i, sk')* from *S*, where |*sk*|=*l*. If there is a record of the form *(sid,pid,P_i pw_i)*, and this is the first New Key query for P_i, then:

1. If this record is **compromised,** or either P_i or P_j is corrupted, then output *(key,sid,pid,sk)* to P_i
2. If this record is **fresh**, and there is a record *(sid,pid,P_j pw_j)* with $pw_j = pw_i$, and a key *sk'* was sent to ,P_j , and *(sid,pid,P_j pw_j)* was **fresh** at the time, then output *(key, sid,pid,sk')* to P_i.
3. In any other case, pick a new random key *sk'* of length *l* and send *(key, sid,pid,sk')* to P_i.

Either way, mark the record *(sid,pid,P_i pw_i)* as **completed**

B Ideal functionality $F_{3\text{-}KD}$

Ideal functionality $F_{3\text{-}KD}$

Trustedserver: Upon receiving a query (*Trustedserver*, *sid*,P_T) from any party P_T, then records (*Trustedserver*, *sid*,P_T) and sends (*Trustedserver*,*sid*,P_T) to all the parties and the adversary *S*.

Initialization: Upon receiving a query (*NewSession*, *sid*,*pid*, P_i,sk_i,*role*) from any party P_i ($P_i \neq P_T$), if *pid*= (P_i,P_j,P_T) for P_T and some party P_j, then record (*sid*,*pid*, P_i,sk_i) and send (*sid*,*pid*,P_i,*role*) to S . Else, ignore this query. Upon receiving a query (*NewSession,sid,pid,*P_T, $sk_i^{'}, sk_j^{'}$,*role*) from party P_T, if *pid*=(P_i,P_j,P_T), then record *(sid,pid,*P_T, $sk_i^{'}, sk_j^{'}$*)* and send (*sid*,*pid*,P_T,*role*) to *S*. If there are already three tuples of the form (*sid*,*pid*,P_i,sk_i), (*sid*,*pid*,P_j,sk_j) and *(sid,pid,*P_T, $sk_i^{'}, sk_j^{'}$*)*, then $F_{3\text{-}KD}$ record (*Session*,*sid*,*pid*,*ready*) and send it to *S*.

Keygeneration: Upon receiving a message (*Session-key*,*sid*,*pid*) from party P_T (for party P_T only), if the tuple (*Session*,*sid*, *pid*,*ready*) has been recorded, then choose $\kappa \xleftarrow{R} \{0,1\}^*$, and record (*Key*,*sid*,*pid*,κ).In addition, if any of P_i , P_j and P_T is corrupted, send (*Key*, *sid*,*pid*,κ) to *S*. Else, send (*Key*,*sid*, *pid*,*key generated*) to *S*.

Keydelivery: 1. Upon receiving a query *(Key,sid,pid,*P_i, κ'*)* from adversary *S*, if the key κ has been generated at that time, then do:

A. If there are records of the form (*sid*,*pid*,P_i,sk_i) and $(sid, pid, P_T, sk_i^{'}, sk_j^{'})$, $sk_i^{'} = sk_i$, both P_T and P_i are not corrupted, then $F_{3\text{-}KD}$ send a private delay output (*Key*,*sid*,*pid*, P_i,κ) to P_i. If either P_i or P_T is corrupted, $F_{3\text{-}KD}$ send (*Key*,*sid*, *pid*,κ) to *S* and send a public delay output (*Key*,*sid*,*pid*, P_i,κ') to P_i .

B. If $sk_i^{'} \neq sk_i$, then if P_i and P_T are uncorrupted, $F_{3\text{-}KD}$ send a public delay output (*Key*,*sid*,*pid*,P_i,*error*) to P_i. Else, send a public delay output (*Key*,*sid*,*pid*,P_i,κ').

2. Upon receiving (*Key*,*sid*,*pid*,P_i,*error*) from the adversary *S* before any key has been already sent to P_i, then send (*Key*,*sid*,*pid*,P_i,*error*) to P_i.