Universally Composable Three-Party Key Distribution

Jin Zhou, TingMao Chang, YaJuan Zhang, YueFei Zhu

***Abstract*—In this paper, we formulate and realize a definition of security for three-party key distribution within the universally composable (UC) framework. That is, an appropriate ideal functionality that captures the basic security requirements of three-party key distribution is formulated. We show that UC definition of security for three-party key distribution protocol is strictly more stringent than a previous definition of security which is termed AKE-security. Finally, we present a real-life protocol that securely realizes the formulated ideal functionality with respect to non-adaptive adversaries.**

***Keywords*—Key distribution, Universally composable, AKE-security**

I. Introduction

Protocols for three-party key distribution allow two client parties in the distributed system to obtain the same session key from a trusted server via the shared private key between each client party and trusted server. These protocols are basic building blocks for contemporary distributed system (eg. three-party key distribution protocol can be used as a modular for constructing three-party key exchange protocol [1]).

Session key distribution in the three-party setting is studied by Needham and Schroeder, which is the trust model assumed by the popular Kerberos authentication system [2]. The provable security for three-party key distribution is provided by Mihir Bellare and Phillip Rogaway by giving the definition of security called AKE-security [3]. (It is emphasized that AKE-security is also an accepted definition of security of other cryptographic tasks, such as group key exchange and key exchange.)

A general framework for representing cryptographic protocols and analyzing their security is proposed by R.Canetti [4, 5]. The framework allows defining the security properties of practically cryptographic tasks. Most importantly, it is shown that protocols proven secure in this framework maintain their security under a very general composition operation, called universal composition, with an unbounded number of copies of arbitrary protocols running concurrently. Similarly, definitions of security formulated in this framework are called universally composable (UC).

The definition of AKE-security follows a definitional approach which is called "security by indistinguishability". In contrast, definitions in the UC framework follow a different definitional approach which is referred to as "security by emulation of an ideal process". In the last few years, researches on the relation between indistinguishability-based definition of security and emulation-based definition of security have become one of the significant topics in cryptography [6]. One case where definitions follow the two approaches were shown to be equivalent is semantically secure encryption against chosen plaintext attacks. However, in most other cases the two approaches result in distinct definitions of security, where the emulation approach usually results in a strictly more restrictive definition. One example, there exists an AKE-secure group key exchange protocol is not UC-secure [7]. Another example, a key exchange protocol is AKE-secure but do not satisfy the emulation-based definition of security [8].

In this paper, we formulate an appropriate ideal functionality that captures the basic security requirements of three-party key distribution, and prove that UC definition of security for three-party key distribution protocol is strictly more stringent than AKE-security. So a real-life protocol which securely realizes the formulated ideal functionality with respect to non-adaptive adversaries is proposed. Therefore, the formulated functionality with security-preserving composition property can be used as a simple building block for modular designs and analysis of complex cryptographic protocols.

II. Preliminaries

Section *A* proposes the description of the distributed system. Next, section *B* reviews the adversarial model. Section *C* sketches the definition of AKE-security. Finally section *D* recalls some writing conventions for ideal functionality.

A. Participants and initialization

For simplicity, we model the distributed system as a fixed polynomial-size set of m client parties $\Pi = \{P_1,\ldots,P_m\}$. 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 secret key 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 distribution protocol at any time (possibly concurrently) in order to obtain a session key. We denote instance s of client party P_i as P_i^s, and denote instance t of trusted server P_T as P_T^t.

B. Adversarial model

In a real attack, the adversary capabilities are modeled

using various oracles. The interaction between an adversary and the protocol participants which is called AKE interaction occurs only via oracle queries. These queries are as follows:

Execute(P_1,s_1,P_j,s_2,P_T,t,*ssid*,*pid*): The output of this query consists of the messages that were exchanged during the honest execution of the protocol with the session (*ssid*,*pid*) among the client instances $P_i^{s_1}$ and $P_j^{s_2}$ and trusted server instance P_T^t.

SendClient(P,s,M): The output of this query is the message that client instance P^s would generate upon receipt of message M.

SendServer(P_T,s,M): The output of this query is the message that server instance P_T^t would generate upon receipt of message M.

StrongCorrupt(P): The output of this query is the internal state of any instances of party P which are concurrently executing the protocol, including the long-term secret key of party P.

Reveal(P,s): It provides the adversary with the session key held by the instance P^s.

Test(P,s): This query does not correspond to any real-world action, but provides a means of defining security. This query is allowed only when P^s has accepted. In response to this oracle call, a random bit b is chosen. If b=0 a random session key is output, while if b=1 the session key of P^s is output. The adversary is allowed to access this oracle only once, at any time during its execution.

C. AKE security

We briefly sketch the definition of AKE-security. A party P is corrupted if the adversary queries **StrongCorrupt**(P). A session (*sid*,*pid*) is corrupted if there exists a $P \in pid$ who is corrupted while there is an instance $\hat{P}^s$ in this session (possibly with $\hat{P} = P$) who has not yet terminated. Say an instance P^s associated with session (*ssid*,*pid*) is **fresh** if the adversary has never queried **Reveal**$(\hat{P}, s')$ for any instance $\hat{P}^{s'}$ in this session, and the session (*ssid*,*pid*) is not corrupted. The adversary *succeeds* (denoted by event **Succ**) if it queries the **Test** oracle regarding a **fresh** instance, and correctly guesses the value of the bit b used by the **Test** oracle in answering this query. Define the advantage of adversary A attacking protocol π to be $Advantage_\pi(A) = \left|\Pr(succ) - 1/2\right|$. Protocol π is said to be AKE-*secure* if, for any poly-time adversary A, the advantage of adversary A is negligible.

D. Some writing conventions for ideal functionality

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**.

III. Universally Composable Three-Party Key Distribution

In this section, we formulate UC definition of security for three-party key distribution and show that any UC-secure three-party key distribution protocol is automatically AKE-secure.

A. Three-party key distribution in the UC framework

The security requirements of a given task are captured via a set of instructions for a "trusted party" that obtains the inputs of the participants and provides them with the desired outputs. We call the algorithm run by the trusted party an **ideal functionality**. Informally, a protocol is said to securely realize the given task if running the protocol amounts to "emulating" an ideal process where the parties hand their inputs to a trusted party with the appropriate functionality and obtain their outputs from it. Thus, to formulate a definition of UC-security for three-party key distribution protocols, we specify an appropriate functionality $F_{3\text{-}KD}$ for three-party key distribution in Figure 1.

The basic idea of $F_{3\text{-}KD}$ 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 now briefly explain the functionality. As expected, the functionality begins with a **trustedserver** phase in which the functionality waits to be notified who is a trusted server and tells all the parties as well as the adversary.

Next, the functionality waits to be notified by two client parties and a trusted server who are supposed to take part in an execution of the protocol. Once $F_{3\text{-}KD}$ receives a notification from the three parties—with identical values of **sid** and **pid**, the functionality enters a "ready" state and sends **ready** message to the adversary.

$F_{3\text{-}KD}$ does not generate the session key until it receives a message (*Sessionkey*,*sid*,*pid)* from P_T. This reflects that in the execution of real-life protocol, the session key should be generated only by P_T. Then $F_{3\text{-}KD}$ chooses a session key

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 (*Sessionkey*,*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*,*keygenerated*) to *S*.

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

1. 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 delayed 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 delayed output (*Key*,*sid*,*pid*, P_i, κ') to P_i .

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

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

κ uniformly at random from $\{0,1\}^k$. At this point, if any of two client parties and trusted server is corrupted, $F_{3\text{-}KD}$ sends κ to the adversary. Else, it sends a message (*Key*,*sid*, *pid*,*keygenerated*) to the adversary. Finally, this session key is delivered to the two client parties. In particular, the session key is delivered to each client party only 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 a **error message** (*Key*,*sid*,*pid*,P_i,*error*). It is emphasized that $F_{3\text{-}KD}$ will send a message (*Key*,*sid*,*pid*,P_i,*error*) to P_i whenever it receives a message (*Key*,*sid*,*pid*,P_i,*error*) from the adversary. This reflects that in the execution of real-life protocol, the adversary can easily cause the error because it can modify and deliver transferred messages at will.

B. Relation to AKE security

We say a three-party key distribution protocol is UC-secure if it securely realizes the ideal functionality $F_{3\text{-}KD}$ .In other words, for any adversary *A*, there exists an ideal adversary *S* such that no PPT environment *Z* can determine whether it is interacting with *A* and parties running with the protocol or interacting with *S* in the ideal process and the ideal functionality $F_{3\text{-}KD}$. Next, we will prove UC definition of security for three-party key distribution is strictly stronger than AKE-security.

Theorem 1 Any UC-secure three-party key distribution protocol is AKE-secure.

Proof Let π be a UC-secure three-party key distribution protocol, and $\hat{\pi}$ be the multi-session extension of π which UC-securely realizes $\hat{F}_{3-KD}$ (the multi-session extension of $F_{3\text{-}KD}$). Assume to the contrary that $\hat{\pi}$ is not AKE-secure. Then there exists an adversary $\hat{A}$ breaking the AKE-security of $\hat{\pi}$ with non-negligible probability. Recall that UC definition of security respects to the "dummy" adversary $\tilde{D}$, we use $\hat{A}$ to construct an environment *Z* so that for any ideal process adversary *S*, *Z* can distinguish whether it interacts with the "dummy" adversary $\tilde{D}$ and parties running $\hat{\pi}$ in the real world, or with *S* and dummy parties communicating with $\hat{F}_{3-KD}$ in the ideal process. Environment *Z* will run $\hat{A}$ as a subroutine, *Z* proceeds as follows:

1. When $\hat{A}$ queries **Execute**(P_1,s_1,P_j,s_2,P_T,*t*,*ssid*,*pid*), *Z* activates a new session with session ID *ssid*, partner ID *pid*. In addition, *Z* records $(P_i^{s_1}, P_T^t, ssid, pid)$ as an uncompleted and unexposed session.

2. When $\hat{A}$ queries **SendClient**(*P*,*s*,*M*), *Z* checks if there is an uncompleted session $(P^s, P_T^t, ssid, pid)$. If not, *Z* returns "invalid query". Otherwise, *Z* delivers *M* to the appropriate session of party *P*, and returns to $\hat{A}$ the response of *P*.

3. When $\hat{A}$ queries **SendServer**(P_T,*t*,*M*), *Z* checks if there is an uncompleted session $(P^s, P_T^t, ssid, pid)$. If not, *Z* returns "invalid query". Otherwise, *Z* delivers *M* to the appropriate session of party P_T, and returns to $\hat{A}$ the response of P_T.

4. When party P outputs (*Key*,*sid*,*pid*,P_i, κ), *Z* records $(P^s, P_T^t, ssid, pid, \kappa)$ as a completed session.

5. When $\hat{A}$ queries **Reveal**(*P*,*s*), *Z* checks if there is a completed session $(P^s, P_T^t, ssid, pid, \kappa)$. If not, *Z* return "invalid query". Otherwise, *Z* gives κ to $\hat{A}$ and marks the

session as exposed.

6. When $\hat{A}$ queries **StrongCorrupt**(*P*), *Z* corrupts party *P* and provides $\hat{A}$ with the internal state of *P*. In addition, *Z* marks *P* and all the uncompleted sessions belonging to *P* as corrupted. Moreover, any sessions invoked by *P* in the future will be marked exposed.

7. When $\hat{A}$ queries **Test**(*P*,*s*), *Z* who checks if there is a record $(P^s, P_T^t, ssid, pid, \kappa)$ marked as completed and unexposed. If not, Z outputs a random bit and halts. Otherwise, *Z* flips a coin $b \leftarrow \{0,1\}$. If *b*=0, Z provides $\hat{A}$ with a random session key. If *b*=1, *Z* provides $\hat{A}$ with κ. Also, *Z* marks the session as **tested**.

8. When $\hat{A}$ outputs a guess bit *b'*, *Z* proceeds as follows:

(a) *Z* finds the **tested** session record $(P^s, P_T^t, ssid, pid, \kappa)$ and then finds all the sessions whose session ID is *ssid* and marks them as **matching.**

(b) *Z* checks if any of sessions marked as **tested** or **matching** is exposed. If it is, *Z* outputs a random bit. Otherwise, *Z* outputs 1 if *b'=b* and outputs 0 if $b' \neq b$.

From the above construction of *Z*, we argue that if *Z* interacts with $\hat{A}$ in the real world, then $\hat{A}$ within *Z* sees in fact an AKE interaction with protocol $\hat{\pi}$. Thus, in this case, *Z* outputs 1 with non-probability better than 1/2. But, when *Z* interacts with *S* and dummy parties in the ideal process, the view of *Z* is statistically independent of *b*. Thus, in this case, *b'=b* with probability exactly one half. Then, it leads to the contradiction that *Z* can distinguish the real world and ideal world with non-probability probability. #

The inverse of this theorem is not held. Note that AKE-security doesn't imply **agreement.** (Our notion of **agreement** requires that any partnered instances (of uncorrupted players) agree on the session key they output [7].) However, it's easy to see that any UC-secure three-party key distribution protocol must guarantee agreement.

IV. UC-SECURE THREE-PARTY KEY DISTRIBUTION PROTOCOL

Since AKE-security doesn't guarantee UC-security, we must propose a real-life protocol that securely realizes $F_{3\text{-}KD}$. The core of our proposed protocol is given in Fig 2.

$$flow1. \quad P_i \rightarrow P_j \quad R_i$$
$$flow2. \quad P_j \rightarrow P_T \quad R_i, R_j$$
$$flow3A. \quad P_T \rightarrow P_i \quad \mu_i, \sigma_i$$
$$flow3B. \quad P_T \rightarrow P_j \quad \mu_j, \sigma_j$$
$$\mu_i = E(sk_i^{enc}, \kappa, r_i) \quad \sigma_i = MAC(sk_i^{mac}, P_i, P_j, R_i, \mu_i)$$
$$\mu_j = E(sk_j^{enc}, \kappa, r_j) \quad \sigma_j = MAC(sk_j^{mac}, P_i, P_j, R_j, \mu_j)$$

Fig 2: the core of our proposed protocol:

Protocol π_I

0. When party P_T is activated with a input (*Trustedserver*, *sid*,P_T), P_T send (*Trustedserver*,*sid*,P_T), to all the parties. Upon receiving a query (*NewSession*,*sid*,*ssid*,*pid*,sk_i,sk_j,*role*), if *role=server*, sk_i, sk_j that has the form of $sk_i = (sk_i^{enc}, sk_i^{mac})$ and $sk_j = (sk_j^{enc}, sk_j^{mac})$, while *pid*= ($P_i$,$P_j$,$P_T$), then P_T does nothing except waiting flow-two message as described below.

1. When party P_i is activated with a input (*NewSession*,*sid*, *pid*,P_i,sk_i,*role*), while *pid*=(P_i,P_j,P_T) and (sk_i^{enc}, sk_i^{mac}), it does as follows. If *role=responder*, it does nothing (except waiting flow-one message as described below), if *role =initiator*, it chooses R_i at random and sends (flow-one,R_i) to P_j. (From this point on, assume that P_i is activated with a query (*NewSession*,*sid*,*ssid*,*pid*,sk_i,*intiator*) and P_j is activated with a query (*NewSession*,*sid*,*ssid*,*pid*,sk_j, *responder*).

2. When P_j receives a message *(*flow-one,R_i*)* from P_i, it chooses R_j at random and sends (flow-two,R_i,R_j) to P_T.

3. When P_T receives a message (flow-two,R_i,R_j) from P_j, it chooses κ, r_i, r_j at random and computes $\mu_i = E(sk_i^{enc}, \kappa, r_i)$, $\mu_j = E(sk_j^{enc}, \kappa, r_j)$, $\sigma_i = MAC(sk_i^{mac}, P_i, P_j, R_i, \mu_i)$ and $\sigma_j = MAC(sk_j^{mac}, P_i, P_j, R_j, \mu_j)$. Then, P_T sends $(\text{flow-three}, \mu_i, \sigma_i)$ and $(\text{flow-three}, \mu_j, \sigma_j)$ to P_i and P_j respectively.

4. When P_i receives a message $(\text{flow-three}, \mu_j, \sigma_j)$ from P_T, it checks if $VF(sk_i^{mac}, \mu_i, P_i, P_j, R_i, \sigma_i) = 1$. If yes, P_i accepts and computes $\kappa = D(sk_i^{enc}, \mu_i)$ and outputs (*Key*,*sid*,*pid*,P_i, κ), else outputs (*Key*,*sid*,*pid*,P_i,*error*). Either way P_i terminates.

When P_j receives a message $(\text{flow-three}, \mu_i, \sigma_i)$ from P_T, it checks if $VF(sk_j^{mac}, \mu_j, P_i, P_j, R_j, \sigma_j) = 1$. If yes, P_j accepts and computes $\kappa = D(sk_j^{enc}, \mu_j)$ and outputs (*Key*,*sid*,*pid*,P_j, κ), else outputs (*Key*,*sid*,*pid*,P_j,*error*). Either way P_j terminates.

Fig 3: The three-party key distribution protocol π_I

The full description of UC-secure three-party key distribution protocol π_I can be found in Fig 3.

V. PROOF OF SECURITY

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

Theorem 2 Assume that (*E, D)* is a CCA-secure symmetric

encryption scheme, that (*MAC, VF*) is EU-CMA-secure message authentication scheme, then three-party key distribution protocol π_1 securely realizes $F_{3\text{-}KD}$ with respect to non-adaptive adversaries.

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_1 in the real world, or with *S* and (dummy) parties communicating with ideal functionality in the ideal process. The description of simulator *S* as follows:

When initialized with the security parameter *k*, the simulator first runs the key generation algorithms of symmetric encryption scheme *E* and message authentication *MAC*, both with security parameter *k*, thus obtaining key pairs $sk_i = (sk_i^{enc}, sk_i^{mac})$ and $sk_j = (sk_j^{enc}, sk_j^{mac})$. Then *S* initializes the real-world adversary *A*. Thereafter, the simulator *S* interacts with the environment *Z*, the ideal functionality $F_{3\text{-}KD}$, and its subroutine *A*. This interaction is implemented by the simulator *S* just following the protocol π_1 on behalf of all the honest parties. The detail of the description of *S* is as follows:

1. *S* generates key $sk_i = (sk_i^{enc}, sk_i^{mac})$ and $sk_j = (sk_j^{enc}, sk_j^{mac})$ on behalf of all parties for protocol π_1.

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 (P_j) is corrupted, *S* sends sk_i' (sk_j') to *A*. If P_T is corrupted, *S* sends the internal state of simulated party P_T together with sk_i', sk_j' to *A*.

3. When *S* receives a message (*Trustedserver*,*sid*,P_T) from $F_{3\text{-}KD}$, *S* sends this message to *A*. When *S* receives a message (*sid*,*pid*,P_i,*role*) from $F_{3\text{-}KD}$, *S* begins simulating for *A* a copy of protocol π_1 (called $\tilde{\pi}_1$) being run by P_i with session ID *sid* and partner ID *pid*. From this now on, any messages sent by *A* to P_i are processed by this simulated copy of $\tilde{\pi}_1$, and any messages output by the simulated copy of $\tilde{\pi}_1$ are given to *A*.

4. Recall the definition of delayed output. If *S* receives a delayed output query which is key delivery message or key error message for some dummy party, it doesn't respond this query until the relevant simulated party in the $\tilde{\pi}_1$ has terminated.

5. When *S* receives a message (*Session*,*sid*,*pid*,*ready*) from $F_{3\text{-}KD}$ (it means the simulated protocol $\tilde{\pi}_1$ also be ready), *S* checks if P_i is corrupted. If not, *S* generates (flow-one, R_i') and sends it to *A* on behalf of P_i. Else, *S* does nothing except waiting flow-one message from *A*.

6. When the simulated party P_j receives (flow-one, R_1) from *A*, *S* checks if P_j is corrupted. If not, *S* generates (flow-two, R_1, R_j') and sends it to *A* on behalf of P_j. Else, *S* does nothing except waiting flow-two message from *A*.

7. When the simulated trusted party P_T receiving (flow-two, R_1,R_2) from *A*,

a) If trusted server P_T is corrupted. *S* does nothing except waiting flow-three message from *A*.

b) Else, if either P_i or P_j is corrupted. In this case, *S* must receive (*Key*,*sid*,*pid*, κ) from $F_{3\text{-}KD}$. Then *S* use κ for generating flow-three messages, Else, *S* chooses a session key κ' randomly for generating flow-three messages. Either way, *S* sends the two generated flow-three messages (flow-three, μ_i', σ_i') and (flow-three, μ_j', σ_j') to *A* on behalf of trusted server P_T.

8. After the simulated party P_i receives a message (flow-three, μ_1, σ_1) 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 $F_{3\text{-}KD}$. Else, if P_i accepts and outputs a session key κ', *S* sends (*Key*,*sid*,*pid*,P_i $D(sk_i, \mu_1)$) to $F_{3\text{-}KD}$.. If P_i outputs error message (*Key*,*sid*,*pid*,P_i, *error*), *S* sends (*Key*,*sid*,*pid*,P_i,*error*) to $F_{3\text{-}KD}$. Then, *S* does similarly for the simulated party P_j.

From the above construction of simulator *S*, it's easy to see that if P_T or P_i (P_j) is corrupted, P_i (P_j) will output the key message or error message as *A* wishes both the real world and the ideal process.

Suppose that no party is corrupted. Then in the real world, *Z* will get no useful information of the session key from each party *P* and the key output by *P* is identical to the random key chosen by trusted server P_T. The first statement hold is attributable to the CCA security of symmetric encryption scheme *E*, while the second statement hold is attributable to 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 process, while P_i will output κ' in the real world iff P_i will output κ in the ideal process. Since both κ' and κ are chosen randomly, so suffice the indistinguishability. (Note that we take the non-adaptive adversary into consideration, the case that occurs in the proof of insecurity of "class" two-move Diffie-Hellman protocol [8] is avoided.)

REFERENCES

[1] 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.

[2] J. Steiner, C. Newman and J. Schiller, Kerberos: "An Authentication Service for Open Network Systems" in Proceedings of the USENIX Winter Conference, pp. 191–202, 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] 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.
[5] R. Canetti: “Universally Composable Security: A New Paradigm for Cryptographic Protocols 2005” in. Revision 3 of ECCC Report TR01-016
[6] 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.
[7] J. Katz, J. Sun Shin. “Modeling Insider Attacks on Group Key-Exchange Protocols” in http://eprint.iacr.org/2005/163
[8] 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.