A Fault-Tolerant Authenticated Key-Conference Agreement Protocol with Forward Secrecy

Tomasz Hyla[1] and Jerzy Pejaś[1]

[1] West Pomeranian University of Technology in Szczecin
Faculty of Computer Science and Information Technology, Poland
{thyla, jpejas}@zut.edu.pl

Abstract. In conference channels, users communicate with each other using a conference key that is used to encrypt messages. There are two basic approaches in which the key can be established. In the first one, a central server is used (with a chairman role). The server generates the key and distributes it to participants. The second approach is that all participants compute a key without a chairman. In this paper, we introduce a special type of group authentication using secret sharing, which provides an efficient way to authenticate multiple users belonging to the same group without the chairman. Our proposed protocol is a many-to-many type of authentication. Unlike most user authentication protocols that authenticate a single user each time, our proposed protocol authenticates all users of a group at once.

Keywords: user authentication; conference-key agreement, group communication, forward secrecy, fault tolerance

1 Introducion

In the era of mobile devices and cloud computing, millions of documents and messages are continuously exchanged over the Internet. The communication channels built using the Internet require security measures that will ensure confidentiality and entities' authentication. Communication channels can be divided into three types: one-to-one, one-to-many (broadcast channels) and many-to-many (conference channels) allowing message exchange between all participants.

Several protocol allowing mutual user authentication and key derivation exists. One of the most popular TLS (Transport Layer Security) is widely used in many kinds of different applications. Situation is more complicated when we need a conference channel. Of course, protocols like TLS can be used to simulate a conference channel by creating independent channels between users. However, when a number of users increases this starts to be infeasible, because the number of channels and exchanged messages increase quadratically.

In conference channels, users communicate with each other using a conference key K that is used to encrypt messages [1]. There are two basic approaches in which the key K can be established. In the first one, a central server is used (with a chairman role).

The server generates the key and distributes it to participants. The second approach is that all participants compute a key without a chairman. The first approach is simpler and easier to implement, but it lacks of flexibility and requires an additional service. The second approach is a special case of a secure multiparty computation in which a group of people evaluate securely a function to $f(k_1, k_2, \ldots)$, with each person possessing a private input k_i. The conference key agreement protocols have a few different features in comparison with secure multiparty computation. Firstly, private channels between participants are not available. Secondly, the main goal of an adversary is to disrupt the protocol between honest participants. Thirdly, a cheater is excluded from participating when is found (the cheater secret is not necessary to compute the conference key) [1].

The conference-key agreement protocols have several properties [2, 3]:

a) Forward confidentiality– *subsequent conference keys cannot be obtained by participants who left the conference sessions* [3, 4].
b) Backward confidentiality – *former conference keys cannot be obtained by participants who joined the conference session* [3, 4].
c) Key freshness – *If a conference-key agreement protocol achieves both backward and forward confidentiality, then the key generated using this protocol is called fresh* [3, 4].
d) Forward secrecy - *A conference-key agreement protocol provides forward secrecy if the long-term secret key x_i of any participant U_i is compromised, but will not result in the compromise of previously established conference keys* [5-7]. Forward secrecy is different from forward confidentiality as it is related to a long-term key in opposite to forward confidentiality that is related to a short term conference key *K*.
e) Authentication – the protocol is called an authenticated protocol when it contains mechanism allowing detect if the participant is a member of the conference [2].
f) Fault tolerance – the protocol is called fault-tolerant when it detects faulty broadcast messages during the communication of participants. The sender of the faulty message is marked as possible malicious one. Faulty messages are re-verified and in a case of a negative result, the participant is excluded from the set of participants [2]. Such property allows honest participant to establish a conference key, even in malicious participant are trying to disrupt it [1] [8-9].
g) Dynamic settings – The dynamic group operations supported by the protocol (i.e.: single or mass join, single or mass leave, merge or divide groups) [2].

Several conference protocols have been proposed in recent years. However, the DKCAP protocol with all mentioned above properties was presented in 2015 by O. Ermiș et al. [2]. Also, they have provided comprehensive comparison of the protocols. The protocol proposed by Chung in 2013 [10] have all the properties except fault tolerance and supports only two dynamic operations. In contrast, Cheng et al. [11] proposed a protocol that is fault tolerant, but does not provide forward secrecy. Zhao et al. [12] proposed protocol that is authenticated, fault tolerant, provides forward secrecy and key freshness, but does not have dynamin operations.

Authenticated, fault tolerant and providing key freshness protocols, without the forward secrecy and dynamin operations have been proposed by Huang et al. [13], Wu et

al. [14], Wang [15]. Katz and Yung [16] proposed an authenticated protocol with forward secrecy and key freshness, but without fault tolerant property and dynamic operations. Tseng proposed two protocol, first one [17] with all the properties except dynamic operations and second one [3] with two dynamic operations, but without key freshness.

1.1 Objective

In this paper, we propose the improved authenticated conference key agreement protocol, called Forward-secrecy and Fault-tolerance Authenticated Conference-Key Agreement (FF-ACKA) protocol. FF-ACKA protocol is a modified version of an improved conference-key agreement protocol with forward secrecy (hereinafter in short ICKAP [2, 17]) for the static group of participants. The protocol is a provable secure conference-key agreement protocol with fault tolerance and forward secrecy. In addition, because the security of ICKAP protocol is preserved, FF-ACKA protocol is resistant against known attacks of a passive and an active adversary.

FF-ACKA protocol requires only a constant size message to be sent by each participant and has two rounds. Both rounds are authenticated by means of long-term certified asymmetric keys of participants. Hence, FF-ACKA protocol is resistant against a classic man-in-the-middle attacks.

Another contribution of this work is to provide a truly contributory group conference key, i.e., in the protocol, the conference key is established jointly by all members of the conference, and not by any single member. Furthermore, the calculation of each participant contribution, in contrast to other protocols (e.g., [2, 17]), is much less time-consuming.

1.2 Paper organisation

The remainder of this paper is organized as follows. Section 2 contains description of a security model, definitions and notations. In Sections 3 is described the proposed FF-ACKA protocol (A Fault-Tolerant Authenticated Key-Conference Agreement Protocol with Forward Secrecy). Next, in Section 4 the security analysis of FF-ACKA protocol is provided. The paper ends with conclusion.

2 Preliminaries

We adopt the system and security model, definitions, and notations similar to Tseng's works ([3], [17], see also [2]).

2.1 System model

In the system, each participant has a long-term private key and a corresponding public key. The public system parameters and each participant's public key information with its certificate are stored in a public directory. All participants are connected using a

broadcast network and can distribute messages to each other. The transmission between them should be protected and needs to establish a conference key to encrypt their communications. Below, we present two definition which is strongly related to FF-ACKA protocol given in Section 3.

Definition 1 (Conference participants, [2]). Participant, participant set and their properties:

(a) Each conference participant is an entity and denoted as U_i.
(b) The participant list is represented as (U_1, U_2, …, , U_n). Each participant knows all participants and theirs identities.
(c) Participants in a conference session can be categorized in two groups. If a participant fully follows the protocol, it is called a honest participant, or if a participant tries to cheat other participants to miscalculate the key is called a malicious participants.

Definition 2 (Verification Matrix, [2]). Let $U = \{U_1, U_2, \ldots, U_n\}$ be the set of participants. During the execution of the protocol, participant U_j, which for $1 \leq i \leq n$ and $i \neq j$ has at least one verification matrix entry $V_{i,j}$ = *'Step4:failure'* or $V_{i,j}$ = *'Step5:failure'* with the proper credentials, is defined as a potential malicious participant until its malicious behaviour is proved in a fault detection and correction step. Otherwise, i.e., if the verification matrix entry $V_{i,j}$ = *'Step4:success''* or $V_{i,j}$ = *'Step5:success''*, the participant is defined as honest participant.

2.2 Security properties

Under the same definitions as used in Tzeng's protocol [1], there are malicious adversaries. The first is a passive adversary (an eavesdropper) who is not a participant, but who tries to learn the conference key from the broadcast messages. The conference-key agreement protocol is secure against a passive attack, if a selected random value and the established conference key are computationally indistinguishable for an adversary. The second is an active adversary. This is an adversary that is also a participant and who tries to disrupt the establishment of a conference key among the honest participants. A conference-key agreement protocol is secure against an active adversary. Even if the adversary does not follow the protocol in any way, he still cannot disrupt the establishment of the conference key. The security of the conference key protocol discussed in the paper is based on the Discrete Logarithm (DL) problem.

3 The Proposed Protocol with Forward Secrecy

Our Forward-secrecy and Fault-tolerance Authenticated Conference-Key Agreement (FF-ACKA) protocol consists of three phases, including a parameter generation phase, long-term keys and a certificates' generation phase, a temporary public-key distribution

phase, a secret distribution and commitment phase, a sub-key computation and verification phase, a fault detection phase, and a conference key computation phase. These phases and theirs detailed structures are explained below.

The FF-ACKA protocol with a forward secrecy property uses a temporary public key to distribute keys, thus even the disclosure of the user's secret key will not result in the compromise of previously established conference keys. The authenticity of the user's temporary key can be verified by checking the signature on the key (hereinafter referred as a temporary certificate) generated with the long-term private key. Hence, each instance of the protocol uses both the static long-term public keys as well as the temporary public keys, which makes the protocol forward secure.

Without loss of generality, let $U = \{U_1, U_2, ..., U_n\}$ be the initial set of participants that want to establish a common conference key. Each U_i ($i = 1, \ldots, n$), knows the set U. In FF-ACKA protocol, we assume that a conference can be started by the creator of U set who as a member of this set plays a role of a conference initiator.

3.1 Step 1: Setup

On input a security parameter $\lambda \in Z$, this algorithm runs as follows (see [18, 19]):

(a) generates a random (λ+1)-bit prime number p;

(b) calculates a prime q number equal to $q = (p - 1)/2$;

(c) chooses an arbitrary $x \in Z_p^*$ with $x \neq \pm 1 \bmod p$ and sets $g = x^2 \bmod p$; g is a generator for the subgroup $G_q \subset Z_p^*$;

(d) chooses cryptographic hash functions (like as in [28]): $H : \{0,1\}^* \to Z_q$, $G : \{0,1\}^* \to Z_q$.

Hence, the system parameters are $params = \{G_q, Z_p, Z_q, p, q, g, H\}$.

3.2 Step 2: Keys and long-term certificates generation

We assume that each user U_i in the system makes the following operations:

(a) picks an random integer $x_i \in Z_q^*$ as a private key and computes a public key y_i such that $y_i = g^{x_i} \bmod p \in Z_p^*$;

(b) sends a certificate signing request to a certificate authority (CA) in order to apply for a digital identity certificate (e.g., in X.509 format) with the U_i's certificate information CI_i, containing the *CA's* identifiers ID_{U_i} and ID_{CA} of the user U_i and the *TA*, respectively, and the time period $\Delta\tau$ for which the information CI_i is valid;

(c) takes his certificate $cert_i$ which was issued by CA.

The resulting public key is (p, q, g, y_i), while the private key has a form of (p, q, g, x_i). The authenticity of this key pair confirms the certificate $cert_i$ issued by the CA. The protocol is started by the initiator who calls for a conference by initializing a set of

participants U. In addition, the function of time stamp T is required, and it will be updated to a new one for each conference session.

3.3 Step 3: Temporary public-key distribution

Each from n participants belonging to a set U generates a temporary key pair and issues certificate related to the key pair. The temporary certificate is signed by the user using his private key and is valid only for a period of the key establishment phase in the conference protocol.

We use the Galindo-Garcia signature scheme given in [20, 21] to generate a temporary certificate. This signature scheme is based on the discrete logarithm problem and is secure against the adaptively chosen message attack in the random oracle model.

The algorithm described below is executed by all users $U_i \in U$:

(a) a user U_i select a short-term private key $t_i \in Z_q^*$ and an integer $v_i \in Z_q^*$ as a private key, and then computes a temporary public key $T_i = g^{t_i} \bmod p$ and $V_i = g^{v_i} \bmod p$;

(b) a user U_i composes the user's certificate information tCI_i, including the U_i*'s* public keys T_i, its identifier ID_i and the short period $\Delta\tau$ for which the information tCI_i is valid;

(c) U_i computes a temporary certificate:

$$tCert_i = \left(t_i + \left(v_i + x_i h_i\right) g_i\right) \bmod q, \tag{1}$$

where $h_i = H\left(tCI_i, V_i, T\right)$ and $g_i = G\left(T_i, T, tCI_i, h_i\right)$ with the U_i*'s* certificate temporary information tCI_i and broadcasts a message tM_i =($tCert_i$, tCI_i, T_i, V_i, T, $\Delta\tau$) to each U group member.

Finally, the $tCert_i$ can be called as a certificate for the temporary public key T_i or a temporary certificate in short.

3.4 Step 4: Secret distribution and commitments

Upon receiving all messages tM_j = ($tCert_j$, tCI_j, T_j, V_j, T, $\Delta\tau$), $j = 1, \ldots, i-1, i+1, \ldots, n$, each user U_i ($1 \le i \le n$) validates that t_j is really issued by U_j, i.e., U_i verifies the time stamp T, computes $h_j = H\left(tCI_j, V_j, T\right)$, $g_j = G\left(T_j, T, tCI_j, h_j\right)$ and checks whether:

$$g^{tCert_j} \equiv T_j \left(V_j \left(y_j\right)^{h_j}\right)^{g_j} \ (mod\ p) \tag{2}$$

The user U_i also validates whether:

(a) the long-term certificate $cert_j$ related to the public key y_j is authentic and unrevoked.

(b) T_j is a generator of a subgroup G_q, i.e., if $2 \le T_j \le p-1$ and $(T_j)^q \bmod p = 1$.

If any check for tM_j of U_j does not hold, then the participant U_i sets $V_{i,j}$ = '*Step4:failure*', otherwise $V_{i,j}$ = '*Step4:success*'. For both cases, the participant U_i computes credentials:

$$h_{k,i} = H\left((T_k)^{x_i} \bmod q, V_{i,j}, T\right),\ k = 1, \ldots, n;\ k \neq i, \tag{3}$$

and broadcasts a tuple $vM_{i,j} = (V_{i,j}, h_{1,i}, \ldots, h_{i-1,i}, h_{i+1,i}, \ldots, h_{n,i}, T)$. Next, if $V_{i,j}$ = '*Step4:failure*', the participant U_i goes to Step 6 for fault detection and correction. Otherwise, each participant U_i:

(a) randomly selects a sub-key $K_i \in Z_q^*$ and a random linear function $f_i(z)$ over Z_q:

$$f_i(z) = K_i + a_i z \in Z_q, \tag{4}$$

where the coefficient a_i is in Z_q;

(b) calculates $\gamma_{i,0} = f_i(1)$, generates random numbers $\alpha_i, \beta_i \in Z_q^*$, calculates $\Gamma_i = g^{\alpha_i} \bmod p$ and $\mathrm{B}_i = g^{\beta_i} \bmod p$, subsequently calculates for each $j = 1, \ldots, n;\ j \neq i$:

(a) $z_{i,j} = (T_j)^{\alpha_i} \bmod q$

(b) $\gamma_{i,j} = f_i(z_{i,j}) \oplus H(z_{i,j}, tCert_i, tCert_j)$

(c) issues an attestation in a form:

$$\sigma_i = (\alpha_i + (\beta_i + t_i c_i) d_i) \bmod q \tag{5}$$

where $c_i = H(\mathrm{B}_i, T)$, and $d_i = G(\Gamma_i, T, K_i, c_i)$; then sends each participant j ($j \neq i$) a message M_i = (σ_i, $cert_i$, $tCert_i$, $\gamma_{i,0}$, $\gamma_{i,1}$, $\gamma_{i,2}$, …, $\gamma_{i,n}$, Γ_i, B_i).

Similarly as in step 3 (see Eq. 2), to issue a certificate a Galindo-Garcia signature scheme [20, 21] was used.

3.5 Step 5: Subkey computation and verification

Upon receiving message M_j = (σ_j, $cert_j$, $tCert_j$, $\gamma_{j,0}$, $\gamma_{j,1}$, $\gamma_{j,2}$, , $\gamma_{j,n}$, Γ_j, B_j) from U_j ($1 \le j \le n$), each participant U_i for $j \neq i$ uses his short-term private key t_i to reconstruct the sub-key K_j as follows:

(a) checks the time stamp T;

(b) computes $z_{j,i} = \left(\Gamma_j\right)^{t_i} \bmod q$ and $f_j\left(z_{j,i}\right) = \gamma_{j,i} \oplus H\left(z_{j,i}, tCert_j, tCert_i\right)$;

(c) calculates a sub-key:

$$K'_j = f_j(1)\frac{z_{j,i}}{z_{j,i}-1} + \frac{-1}{z_{j,i}-1} f_j\left(z_{j,i}\right) \in Z_q \tag{6}$$

(d) checks if the following equality is fulfilled:

$$g^{\sigma_j} = \Gamma_j\left(\mathrm{B}_j\left(T_j\right)^{c'_j}\right)^{d'_j} (mod\ p) \tag{7}$$

where $c'_j = H\left(\mathrm{B}_j, T\right)$ and $d'_j = G\left(\Gamma_j, T, K'_j, c'_j\right)$.

If it holds, a user U_i accepts a sub-key $K_j = K'_j$ and sets $V_{i,j}$ = '*Step4:success*'. Otherwise, a participant U_i sets $V_{i,j}$ = *'Step5:failure'*. For both cases the participant U_i computes the credentials:

$$r_{k,i} = H\left(z_{k,i}, V_{i,j}, T\right),\ k = 1, \ldots, n;\ k \neq i, \tag{8}$$

and broadcasts a tuple $rM_{i,j} = (V_{i,j}, r_{1,i}, \ldots, r_{i-1,i,}, r_{i+1,i}, \ldots, r_{n,i}, T)$.

3.6 Step 6: Fault detection

All faults messages broadcasted by participants in steps 4 and 5 should be verified and corrected. Each participant U_j, which does not follow the protocol in any way, should be treated as possible malicious participants and marked as $V_{i,j}$ = '*StepX:failure*' in the verification matrix, where $X = 4$ or 5. Such approach is reasonable, because any malicious participant U_j can try to send the wrong evidences to other participants and disrupt the establishment of a conference key among the honest participants. The broadcast messages of possible malicious participants are re-verified by honest participants.

The broadcast messages of participants are verified in both secret distribution and commitment step (Step 4) and the sub-key computation and verification step (Step 5). Therefore, the fault messages in the FF-PA protocol that are sent by any malicious participant to other participants should be re-verified by honest ones.

Before the proper fault detection phase is started, each participant U_i ($1 \le i \le n$) on receiving the message $vM_{j,m}$ or/and $rM_{j,m}$ (see respective Step 4 and Step 5), first computes (for $i \neq j \neq m$):

$$h'_{i,j} = H\left(\left(y_j\right)^{t_i} \bmod q, V_{j,m}, T\right), \tag{9}$$

or/and

$$r'_{i,j} = H\left(\left(\Gamma_j\right)^{t_i} \bmod q, V_{j,m}, T\right), \tag{10}$$

and then verifies whether holds the equations $h'_{i,j} \equiv h_{i,j}$ or/and $r'_{i,j} \equiv r_{i,j}$. If the equation(s) are satisfied, then U_i starts the execution of the fault detection phase, or else,

sets U_j as a malicious participant and restarts the protocol for new honest participant group $U = U \setminus \{U_i\}$.

To detect and correct the faults, each participant $U_i \in U$ ($i \neq j \neq m$) receiving $V_{j,m}$ acts in accordance to the following rules (compare also [2, 13, 22]):

(a) if $V_{j,m}$ = *'Step4failure'* for any participant U_m, then other honest participants U_l, where $1 \leq l \leq n$, re-verify the broadcast messages tM_m and if the verification result is still $V_{l,m}$ = *'failure:Step4'* (for any participant U_l), the participant U_l sets U_m as a malicious participant and goes to the step (c);

(b) otherwise, if $V_{j,m}$ = *'Step5:failure'*, the participants U_i knows that according to U_j the source of faulty is U_m ($m \neq i$); hence U_i:

(i) waits for the fault detection message (α_m, a_m, K_m) from U_m;

(ii) sets U_m as a malicious participant and goes to the step (c), if no one receives the fault detection message from U_m;

(iii) checks whether previously distributed values (V_m, T_m, $tCert_m$, $\gamma_{m,0}$, $\gamma_{m,j}$, Γ_m, B_m, σ_m) are correct, i.e.:

- checks whether α_m satisfies $\Gamma_m \equiv g^{\alpha_m} \bmod p$;
- checks whether Eq. (1) holds for V_m, T_m and $tCert_m$;
- inputs (a_m, K_m) into Eq. (4), calculates $z_{m,j} = \left(T_j\right)^{\alpha_m} \bmod q$ and checks whether:

$$\gamma_{m,0} \equiv f_m(1)$$
$$\gamma_{m,j} \equiv f_m\left(z_{m,j}\right) \oplus H\left(z_{m,j}, tCert_m, tCert_j\right)$$

- verifies whether (Γ_m, B_m, σ_m) is the right signature on a sub-key K_m made by U_m (see Eq. (7)); if Eq. (7) holds, sets U_j as a malicious participant, otherwise, the malicious one is U_m.

(c) removes the malicious participant (U_i or U_m) from the group U, i.e., the set of users is updated as $U = U \setminus \{U_j\}$; similar operations are made by the other honest participants and finally the protocol is restarted.

3.7 Step 7: Conference-key computation

If no more faults are detected and all malicious participants are excluded from U, each honest participants U_i in the set of $U' = \left\{U_{i,1}, U_{i,2}, \ldots, U_{i,m}\right\}$, where $m \leq n$, may calculate the conference key K as follows:

$$K = \left(K_{i,1} + K_{i,2} + \ldots + K_{i,m}\right) \bmod q \tag{11}$$

4 Security Analysis

In this section, we give the security analysis of FF-ACKA protocol in fault tolerance and its forward secrecy. In the paper, a formal security analysis of the protocol is omitted. The formal analysis includes protocol resistance to passive and active attacks. Because of the similarity of the structures of the FF-ACKA and DCKAP [2] protocols, the security proof is similar to analysis presented in [2, 3].

For the fault tolerance analysis of FF-ACKA protocol we use the same approach as given in [1, 2, 13]. Let's consider two general attack scenarios on FF-ACKA protocol (compare also [22]). In the first scenario, all participants may generate different conference keys, while in second one the honest participants can be excluded from the conference. Furthermore, we assume that an adversary A is able to alter the exchanged messages in the sub-key computation and verification phase.

Given the first scenario, the participant U_i is a malicious participant who sends a wrong message tM_j (Step 4) or M_j (Step 5). Suppose, that one of the honest participants U_j broadcasts message $V_{j,i}$='*Step4:failure'* or $V_{j,i}$='*Step5:failure'* after verifying the digital signature $tCert_j$ (Eq. (3)) or σ_i (Eq. (8)). However, an adversary A modifies the intercepting message $V_{j,i}$='*Step4:failure'* or $V_{j,i}$='*Step5:failure'* to $V_{j,i}$='*Step4:success'* or $V_{j,i}$='*Step5:success'* and broadcasts it to other participants. As a result, all participants compute and accept different conference keys.

For the second scenario, both U_i and U_j are honest participants in the set U. In Step 4 or Step 5 the participant U_j verifies receiving message tM_j (Step 4) or M_j (Step 5). If check holds, U_j broadcasts $V_{j,i}$='*Step4:success'* or $V_{j,i}$='*Step5:success'*. However, like as in the first scenario, any adversary A can intercept this message and modify it to $V_{j,i}$='*Step4: failure'* or $V_{j,i}$='*Step5: failure'*. It is obvious that all participants start the fault detection procedure and although the participants U_i is honest all participants remove it from the set of honest participants. Of course participant U_i should be removed from a set of honest participants, but only if U_i is indeed a malicious participant and sends out the fake message $V_{i,j}$='*Step4: failure'* or $V_{i,j}$='*Step5: failure'*, declaring U_j who is actually honest as a malicious one.

In the following theorem, we demonstrate that the proposed protocol can provide fault tolerance.

Theorem 1 (Fault tolerance, [1, 2, 22]). If honest participants follow the protocol, they may compute the same conference key even if:

(i) A malicious participant cheats the honest participants by sending wrong key values, and
(ii) A malicious participant cheats the honest participants by identifying an honest participant as a possible malicious participant.

Proof. For the first condition, assume that the participant U_i is a malicious participant and attempts to disrupt the establishment of a conference key among honest participants. The malicious participant U_i can use two methods of attacks. In first one, U_i broadcasts wrong messages in Step 4 or Step 5, while in second one U_i broadcasts the

message $V_{i,j}$='*Step4: failure'* or $V_{i,j}$='*Step5:failure'* to claim that U_j is a malicious participant, although U_j is indeed a honest participant.

If U_i tries to cheat other honest participants in Step 4, then the broadcast messages $tM_i = (tCert_i, tCI_i, T_i, V_i, T, \Delta\tau)$ are validated by each honest participant by checking whether $g^{tCert_i} \equiv T_i\left(V_i (y_i)^{h_i}\right)^{g_i} \pmod p$, where $h_i = H(tCI_i, V_i, T)$, $g_i = G(T_i, T, tCI_i, h_i)$, holds or not. If the message tM_i of U_i is wrong, at least one honest participant U_j ($j \neq i$) is able to claim $V_{j,i}$='*Step4:failure'*, because the wrong messages tM_i are unable to pass the validation. If U_i broadcasts wrong messages M_i in Step 5, each honest participant U_j can also validate the message M_i and concludes if the equations $g^{\sigma_i} = \Gamma_i\left(\mathrm{B}_i (T_i)^{c_i'}\right)^{d_i'} \pmod p$, where $c_i' = H(\mathrm{B}_i, T)$ and $d_i' = G(\Gamma_i, T, K_i', c_i')$, holds or not. If the K_i' value is wrong the validations do not hold, and so any honest participant U_j can claim that $V_{j,i}$='*Step5:failure'*.

Additionally, for both above presented forgeries assume that one of the honest participant U_j broadcasts message $V_{j,i}$ = '*Step4: failure'* or $V_{j,i}$ = '*Step5:failure'* after verifying the wrong message tM_i or M_i. However, an adversary A can intercept the message $vM_{j,i}$ or $rM_{j,i}$ sent by U_j and modify this message to $V'_{j,i}$ = '*Step4:success'* or $V'_{j,i}$ = '*Step5:success'*, and then broadcasts it to all other participants.

Let's take a closer look at above described adversary attack. It is easy to see that this type of attack is detected in the early stage of the fault detection phase. To show this, suppose that an adversary A replaces $V_{j,i}$ with $V'_{j,i}$ and resends it to all other participants. On receiving $V'_{j,i}$ and $h_{k,j}$ (Eq. (4)) or $r_{k,j}$ (Eq. (8)), any U_k ($k \neq i \neq j$) first computes $h'_{k,j} = H\left(\left(y_j\right)^{t_k} \bmod q, V'_{j,i}, T\right)$ or $r'_{k,j} = H\left(z_{k,j}, V'_{j,i}, T\right)$ and checks whether $h'_{k,j} \equiv h_{k,j}$ or $r'_{k,j} \equiv r_{k,j}$ holds or not. It is obvious that if these equations holds, an adversary should be able to compute $w_{k,j} = \left(T_k\right)^{x_j} \bmod q$ or $z_{k,j} = \left(\Gamma_j\right)^{t_k} \bmod q$. However, it is well known that a problem of finding $w_{k,j}$ or $z_{k,j}$ values is computationally hard, i.e., to obtain these values we need to solve DL problem for the equations $y_j = g^{x_j} \bmod p$ and $T_k = g^{t_k} \bmod p$ or $\Gamma_j = g^{\alpha_j} \bmod p$.

In the second method of attacks, a malicious participant U_i broadcasts the message $V_{i,j}$='*Step4: failure'* or $V_{i,j}$='*Step5:failure'* to claim that U_j is a malicious participant, while U_j is actually an honest participant. In such case, U_j resends message tM_j (for Step 4) or M_j and (α_j, a_j, K_j) (for Step 5) to prove his honesty. Next, other honest participants validate message tM_j by checking $g^{tCert_j} \equiv T_j\left(V_j \left(y_j\right)^{h_j}\right)^{g_j} \pmod p$ (see point (a) of Step 6) or messages M_j and (α_j, a_j, K_j) according to the point (b) of Step 6.

The information about any malicious attempt of U_i or an adversary A who eavesdrops on the broadcast channel and tries to alter the exchanged messages in Step 4 or Step 5, is placed in $V_{j,i}$ element of the verification matrix, for all $1 \leq j \leq n$, j ≠ i. On receiving such fault messages, all honest participants start the fault detection phase and decide

that U_i is indeed malicious participant. As a result, a participant U_i will be proved to be the malicious participant and then be removed from the set of honest participants.

Now, consider second condition (ii). Since in this case, where any honest participants U_i and U_j follow the protocol and broadcast valid messages, all participants can properly compute the same sub-key K_i. This is truth even if a malicious participant U_m sends out the failure message to convince other honest participants that U_i and U_j are dishonest or if an adversary A intercepts the message $V_{i,j}$ = '*Step4:success*' or $V_{i,j}$ = '*Step5:success*' sent from U_i and changes it to $V_{i,j}$ = '*Step4: failure*' or $V_{i,j}$ = '*Step5: failure*'. It can be noticed, that as above, these attacks are detected in fault detection phase.

Because these two conditions hold, our FF-ACKA protocol satisfies the fault-tolerance property. The last security property in this section is the forward secrecy. Below, we demonstrate that the proposed protocol has the forward secrecy property. Forward secrecy should protect the previously established conference keys and all its components against compromises, even if the long-term private key is compromised.

Theorem 2 (Forward secrecy, [17]) If solving the discrete logarithm problem is computationally infeasible, the FF-CKAP provides forward secrecy.

Proof. The proof of Lemma 2 is similar to the proof of [17]. Suppose that adversary A compromises at time $(\tau + 1)$ the long-term private key x_i of any participant U_i and can obtain the conference key K at time τ, where K is composed of all participants' sub-keys K_i ($1 \le i \le n$). Because each K_i is distributed to other participants U_j ($1 \le j \le n, j \ne i$) using their temporary public keys $T_j = g^{t_j} \bmod p$, an adversary A should be able to obtain the short-term private key t_j. The adversary can try to get t_j directly from $T_j = g^{t_j} \bmod p$ or indirectly either form $tCert_j = \left(t_j + \left(v_j + x_j h_j\right) g_j\right) \ mod \ q$ or $\sigma_j = \left(\alpha_j + \left(\beta_j + t_j c_j\right) d_j\right) \ mod \ q$ (see respective Eq. (1) and Eq. (4)). For the first case, it is obvious that the calculation of t_j is equivalent to solving the discrete logarithm problem. However, DL problem is nevertheless considered to be computationally hard. For the second case, even if the adversary knows x_j, the solving of $tCert_j = \left(t_j + \left(v_j + x_j h_j\right) g_j\right) \ mod \ q$ to obtain t_j is also discrete problem [20]. In the last case, the adversary with U_j*'s* secret key x_j tries to recover t_j form (T_j, Γ_j, B_j, σ_j), where $\Gamma_i = g^{\alpha_i} \bmod p$, $B_i = g^{\beta_i} \bmod p$ and $\sigma_j = \left(\alpha_j + \left(\beta_j + t_j c_j\right) d_j\right) \ mod \ q$. The value σ_j (like $tCert_j$) is short signature computed using a signature scheme considered in [20]. According to Theorem 1 in [20] this signature scheme is existentially unforgeable under a chosen message attack in the random oracle model, assuming that DL problem is believed to be computationally hard. As a result, the proposed protocol FF-CKAP satisfies the forward secrecy under the difficulty of computing the discrete logarithm problem.

5 Conclusion

In this paper, we proposed an improved authenticated conference-key agreement protocol (FF-ACKA) for a static set of participants. The protocol provides important security properties like forward secrecy and fault-tolerance. Due to the last properties it is possible to ensure that all participants can obtain the same conference key. The proposed protocol uses only two rounds to generate a conference key and enables the efficient detection and elimination of malicious participants from the set of honest participants. Nevertheless, the size of messages exchanged during protocol is proportional to the number of participants.

The FF-ACKA protocol was developed to meet requirements that emerged while we have been developing MobInfoSec project [23-27]. We have needed an authenticated conference channel between mobile devices. We have achieved such channel using several one-to-one channels with a chairman. Such solution was working, but it was not scalable and generated some efficiency and security issues related to the service implementing a chairman role. The FF-ACKA protocol is designed to solve that two problems: to improve overall speed and to improve security by eliminating the necessity to have trust service that plays a chairman role. Preliminary analysis has shown that our protocol should have similar efficiency or in some situations should be faster than protocols proposed by [1, 2, 16, 22]. We are planning to implement the protocol during our future works and test it in several practical scenarios.

References

1. Tzeng, W.G.: A secure Fault-Tolerant Conference-Key Agreement Protocol. IEEE Transactions on Computers, vol. 51, no. 4, 373-379 (2002)
2. Ermiş, O., Bahtityar, S., Anarim, E., Çağlayan, M.U.: An improved conference-key agreement protocol for dynamic groups with efficient fault correction. Security and Communication Networks, vol. 8, issue 7, 1347-1359 (2015)
3. Tseng, Y.M.: A communication-efficient and fault-tolerant conference-key agreement protocol with forward secrecy. The Journal of Systems and Software, vol. 80, issue 7, 1091-1101 (2007)
4. Rhee, K.H., Park, Y.H., Tsudik, G.:An architecture for key management in hierarchical mobile ad-hoc networks. J. Commun. Networks 6 (2), 1–7 (2004)
5. Bellare, M., Pointcheval, D., Rogaway, P.: Authenticated key exchange secure against dictionary attacks. In: Preneel, B. (eds.) Proc. Advances in Cryptology – Eurocrypt'00, pp. 139–155. Springer Heidelberg (2000)
6. ANSI X9.63.: Public key cryptography for the financial services industry: key agreement and key transport using Elliptic Curve cryptography. ANSI (2001)
7. Tseng, Y.M.,: A robust multi-party key agreement protocol resistant to malicious participants. Comput. J. 48 (4), 480–487 (2005)
8. Katz, J., Shin, J.S.: Modelling insider attacks on group key exchange protocols. In: ACM Conf. on Computer and Communications Security, pp. 180–189 (2005)
9. Tang, Q., Mitchell, C.J.: Security properties of two authenticated conference key agreement protocols. In: Information and Communications Security: 7th Int. Conf., ICICS 2005, LNCS 3783, pp. 304–314 (2005)

10. Chung, YF.: The design of authentication key protocol in certificate-free public key cryptosystem. Security Communication Networks. vol. 7, issue 11, 2125-2133 **(**2013)
11. Cheng, ZY., Liu Y., Chang CC., Guo, C.: A fault-tolerant group key agreement protocol exploiting dynamic setting. International Journal of Communication Systems. vol. 26, issue 2, 259–275 (2013)
12. Zhao, J., Gu, D., Li ,Y.: An efficient fault-tolerant group key agreement protocol. Computer Communications. vol. 33, 890–895 (2010)
13. Huang, K.H., Chung, Y.F., Lee, H.H., Lai, F., Chen, T.S.: A conference key agreement protocol with fault-tolerant capability. Computer Standards & Interfaces. vol. 31, no. 2, 401-405 (2009)
14. Wu, Q., Mu, Y, Susilo, W., Qin, B., Domingo-Ferrer, J.: Asymmetric group key agreement. In: Advances in Cryptography - EUROCRYPT 2009, LNCS, vol. 5479, pp. 153–170. Springer Heidelberg (2009)
15. Wang, Z.: Improvement on the fault-tolerant group key agreement protocol of Zhao et al. Security Communication Networks. vol. 9, issue 2, 166-170 (2016).
16. Katz, J., Yung, M.: Scalable protocols for authenticated group key exchange. Journal of Cryptology. vol. 20, issue 1, 85–113 (2007)
17. Tseng, Y.M.: An improved conference-key agreement protocol with forward secrecy. Informatica. vol. 16, 275–284. Lithuania Academy of Sciences (2005)
18. Ryabko, B., Fionov, A.: Basics of Contemporary Cryptography for IT Practioners. World Scientific Publishing Co Pte Ltd (2005)
19. Katz, J., Lindell, Y.: Introduction to Modern Cryptography: Principles and Protocols. Chapman and Hall/CRC (2007)
20. Chatterjee, S., Kamath, Ch., Kumar V.: Galindo-Garcia Identity-Based Signature Revisited. In Kwon, T., Lee, M.-K., Kwon, D. (Eds.), Information Security and Cryptology – ICISC 2012, LNCS, vol. 7839, pp. 456-471. Springer Heidelberg (2013)
21. Chatterjee, S., Kamath, Ch.: A Closer Look at Multiple Forking: Leveraging (In)Dependence for a Tighter Bound. Algorithmica. vol. 74, issue 4, 1-42 (2015)
22. Lee, C.C., Li, C.T., Wu, C.Y., Huang, S.Y.: An Enhanced Fault-Tolerant Conference Key Agreement Protocol. International Journal of Computer, Electrical, Automation, Control and Information Engineering. vol. 8, no. 12, 2231 – 2235 (2014)
23. El Fray, I., Hyla, T., Kurkowski, M., Maćków, W., Pejaś, J.: Practical authentication protocols for protecting and sharing sensitive information on mobile devices. In: Kotulski, Z., Księżopolski, B., Mazur, K. (eds.) CSS 2014. CCIS, vol. 448, pp. 153–165. Springer, Heidelberg (2014)
24. El Fray, I., Hyla, T., Chocianowicz, W.: Protection Profile for Secure Sensitive Information System on Mobile Devices. In: Saeed, K., Snášel, V. (eds.) CISIM 2014. LNCS, vol. 8838, pp. 636–650. Springer, Heidelberg (2014)
25. Hyla, T., Pejaś, J.: Certificate-Based Encryption Scheme with General Access Structure. In: Cortesi, A., Chaki, N., Saeed, K., Wierzchoń, S. (eds.) CISIM 2012. LNCS, vol. 7564, pp. 41–55. Springer-Verlag, Berlin (2012)
26. Hyla, T., Pejaś, J.: A practical certificate and identity based encryption scheme and related security architecture. In: Saeed, K., Chaki, N., Cortesi, A., Wierzchon, S. (eds.), CISIM 2013. LNCS, vol. 8104, pp. 178-193. Springer-Verlag, Berlin (2013)
27. Hyla, T., Maćków, W., Pejaś, J.: Implicit and explicit certificates-based encryption scheme. In: Saeed, K., Snášel, V. (eds.) CISIM 2014. LNCS, vol. 8838, pp. 651–666. Springer, Heidelberg (2014)
28. IEEE Standard 1363.3 – 2013 – IEEE Standard for Identity-Based Cryptographic Techniques using Pairings, (2013)