Efficient Compilers for Authenticated Group Key Exchange

Qiang Tang and Chris J. Mitchell
Information Security Group
Royal Holloway, University of London
Egham, Surrey TW20 0EX, UK
{qiang.tang,c.mitchell}@rhul.ac.uk

10th October, 2005

Abstract

In this paper we propose two compilers which are designed to transform a group key exchange protocol secure against any passive adversary into an authenticated group key exchange protocol with key confirmation which is secure against any passive adversary, active adversary, or malicious insider. We show that the first proposed compiler gives protocols that are more efficient than those produced by the compiler of Katz and Yung. The second proposed compiler further reduces the computational complexity of the output protocols by using a Trusted Third Party (TTP). We moreover show that, although the protocols produced by the novel compilers have lower computational complexity than the protocols produced by the Katz-Yung compiler, the protocols nevertheless achieve key confirmation, unlike the protocols output by the Katz-Yung compiler.

1 Introduction

In recent years authenticated group key exchange protocols have received increasing attention. Authenticated group key exchange enables a group of participants to establish a common secret key over insecure public networks, even when adversaries completely control all the communications. Authenticated group key exchange ensures that only the intended users can possibly compute the session key.

The case of 2-party authenticated key exchange has been well investigated within the cryptographic community, and a variety of efficient and provably-secure protocols have been proposed (see e.g. [1, 4, 5, 6, 11, 12]). However, less attention has been given to the case of authenticated group key exchange protocols, which have more than two participants. A number of authors have considered extending the 2-party Diffie-Hellman protocol [14] to the group setting (see e.g. [10, 15, 18, 21]). Unfortunately, most of these schemes only assume a passive adversary, so they are vulnerable to active adversaries who control the communication network. Recently Bresson et al. [9] took the lead in presenting a formal model of security for authenticated group key exchange and providing rigorous proofs of security for particular protocols. Their model builds on earlier work of Bellare and Rogaway in the 2-party setting [5]. Following Bresson et al.'s work, several provably secure authenticated group key agreement protocols (see e.g. [8, 13, 17]) have been proposed.

In this paper we are particularly interested in compilers which transform a protocol of one type into another. In [2] Bellare, Canetti, and Krawczyk presented a compiler which transforms unauthenticated protocols into authenticated protocols in the 2-party setting. In [19], Mayer and Yung give a compiler which transforms any 2-party protocol into a centralised group protocol which, however, is not scalable. Recently, Katz and Yung [17] proposed a compiler (referred to as the Katz-Yung compiler) which transforms a group key exchange protocol secure against any passive adversary into an authenticated group key exchange protocol which is secure against both passive and active adversaries. The security of the Katz-Yung compiler is rigorously proved, and the protocols produced by this compiler are also more efficient and scalable than those produced by the method given in [19]. Although the Katz-Yung compiler produces more efficient protocols than the Mayer-Yung compiler, it nevertheless still produces rather inefficient protocols. Each participant is required to perform numbers of signatures and verifications proportional to the number of rounds in the original protocol and the number of participants involved, respectively. Additionally, the Katz-Yung compiler adds an additional round to the original protocol, but does not achieve key confirmation.

In this paper we show that, if a specific group key exchange protocol is used as input, the authenticated protocol created by the Katz-Yung compiler can be simplified without any loss of security. Based on our analysis, we propose two new, more efficient, compilers and prove their security. Both our new compilers result in protocols achieving key confirmation with lower computational complexity and round complexity than those produced by the compiler of Katz and Yung.

The rest of this paper is organised as follows. In section 2, we review the Katz-Yung compiler, and give a performance analysis. We also give an example of the use of the compiler and show how to simplify the protocol produced. In section 3, we propose a new compiler which transforms a group key exchange protocol secure against any passive adversary into an authenticated group key exchange protocol with key confirmation which is secure against any passive adversary, active adversary, or malicious insider. In section 4, we propose a second new compiler which outputs protocols that are both more efficient and provide the same functionality as the first compiler, at the cost of introducing a TTP. In section 5, we conclude the paper.

2 The Katz-Yung compiler

In this section we first review the Katz-Yung compiler and also discuss the security and efficiency of the protocols it produces. We then describe an example of an authenticated key agreement protocol output by this compiler. Finally, we show how to simplify this authenticated key agreement protocol.

In this paper, with respect to a protocol P input to the compiler, which must be a group key exchange protocol secure against any passive adversary, we make the following assumptions:

1. There is no key confirmation, and all participants compute the session key after the last round of the protocol.
2. Every protocol message is transported together with the identifier of the source and the round number.

2.1 Description of the Katz-Yung compiler

Let $\Sigma = (Gen, Sign, Vrfy)$ be a signature scheme which is strongly unforgeable under an adaptive chosen message attack (see, for example, [3] for a definition). If k is a security parameter, then $Gen(1^k)$ generates a pair of public/private keys for the signing and verification algorithms ($Sign$, $Vrfy$).

Suppose a set $S = \{U_1, \cdots, U_n\}$ of users wish to establish a session key. Let ID_{U_i} represent U_i's identity for every i ($1 \leq i \leq n$). Given P, a group key exchange protocol secure against any passive adversary, the compiler constructs a new protocol P', in which each party $U_i \in S$ performs as follows.

1. In the initialisation phase, and in addition to all the operations in protocol P, each party $U_i \in S$ generates a verification/signing key pair (PK_{U_i}, SK_{U_i}) by running $Gen(1^k)$, where k is a security parameter.

2. Each user U_i chooses a random $r_i \in \{0,1\}^k$ and broadcasts $ID_{U_i}||0||r_i$, where here, as throughout, $||$ represents concatenation. After receiving the initial broadcast message from all other parties, each U_i sets $nonce_i = ((ID_{U_1}, r_1), \cdots, (ID_{U_n}, r_n))$ and stores this as part of its state information.

   It is obvious that all the users will share the same nonce, i.e., $nonce_1 = nonce_2 = \cdots = nonce_n$, as long as no attacker changes the broadcast data (or an accidental error occurs).

3. Each user U_i in S executes P according to the following rules.

   - Whenever U_i is supposed to broadcast $ID_{U_i}||j||m$ as part of protocol P, it computes $\sigma_{ij} = Sign_{SK_{U_i}}(j||m||nonce_i)$ and then broadcasts $ID_{U_i}||j||m||\sigma_{ij}$.
   - Whenever U_i receives a message $ID_U||j||m||\sigma$, it checks that: (1) $U \in S$, (2) j is the next expected sequence number for a message from U, and (3) $Vrfy_{PK_U}(j||m||nonce_i, \sigma) = 1$ where here, as throughout, 1 signifies acceptance. If any of these checks fail, U_i aborts the protocol. Otherwise, U_i continues as it would in P upon receiving message $ID_U||j||m$.

4. Each non-aborted protocol instance computes the session key as in P.

2.2 Security and efficiency

Katz and Yung [17] claim that their proposed compiler provides a scalable way to transform a key exchange protocol secure against a passive adversary into an authenticated protocol which is secure against an active adversary. They also illustrate efficiency advantages over certain other provably-secure authenticated group key exchange protocols. With respect to efficiency, we make the following observations on the protocols produced by the Katz-Yung compiler.

1. Each user U_i must store the nonce $nonce_i = ((U_1, r_1), \cdots, (U_n, r_n))$ regardless of whether or not the protocol ends successfully. Since the length of this information is proportional to the group size, the storage of such state information will become a non-trivial overhead when the group size is large.

2. From the second round onwards, the compiler requires each user to sign all the messages it sends in the protocol run, and to verify all the messages it receives. Since the total number of signature verifications in one round is proportional to the group size, the signature

verifications will potentially use a significant amount of computational resource when the group size is large.

3. The compiler adds an additional round to the original protocol P; however, it does not provide key confirmation. As Katz and Yung state [17], in order to achieve key confirmation a further additional round is required.

2.3 An example application of the compiler

The following authenticated group key exchange protocol is obtained by applying the compiler to the protocol proposed by Burmester and Desmedt [10]. The resulted authenticated protocol has been proven secure by Katz and Yung [17].

Suppose a cyclic group G with prime order q and generator $g \in G$ is determined in advance and known to all the users. Suppose a set $S = \{U_i, \cdots, U_n\}$ of users wish to establish a session key. Let ID_{U_i} represent U_i's identity for every i $(1 \leq i \leq n)$. During the initialisation phase, each user $U_i \in S$ generates the verification/signing keys (PK_{U_i}, SK_{U_i}) by running $Gen(1^k)$. The protocol operates as follows.

1. Each user U_i begins by choosing a random $r_i \in \{0,1\}^k$ and broadcasting $U_i||0||r_i$. After receiving the initial broadcast message from all other parties, U_i sets $nonce_i = ((U_1, r_1), \cdots, (U_n, r_n))$ and stores it as part of its state information.

2. Each user U_i chooses a random $s_i \in Z_q$ and computes $Z_i = g^{s_i}$. U_i computes a signature $\sigma_{i1} = Sign_{SK_{U_i}}(1||Z_i||nonce_i)$ and then broadcasts $ID_{U_i}||1||Z_i||\sigma_{i1}$.

3. When U_i receives any message $ID_{U_j}||1||Z_j||\sigma_{j1}$ from user U_j $(1 \leq j \leq n, j \neq i)$, it checks that: (1) $U_j \in S$, (2) 1 is the next expected sequence number for message from U_j, and (3) $Vrfy_{PK_{U_j}}(1||Z_j||nonce_i, \sigma_{j1}) = 1$. If any of these checks fail, U_i aborts the protocol. Otherwise, U_i computes $X_i = (Z_{i+1}/Z_{i-1})^{r_i}$ and the signature $\sigma_{i2} = Sign_{SK_{U_i}}(2||X_i||nonce_i)$, and broadcasts $ID_{U_i}||2||X_i||\sigma_{i2}$.

4. When U_i receives any message $ID_{U_j}||2||X_j||\sigma_{j2}$ from user U_j $(1 \leq j \leq n, j \neq i)$, it checks that: (1) $U_j \in S$, (2) 2 is the next expected sequence number for message from U_j, and (3) $Vrfy_{PK_{U_j}}(2||X_j||nonce_i, \sigma_{j2}) = 1$. If any of these checks fail, U_i aborts the protocol.

U_i computes the session key as:

$$\begin{aligned} K_i &= (Z_{i-1})^{ns_i} \cdot (X_i)^{n-1} \cdot (X_{i+1})^{n-2} \cdots X_{i+n-2} \\ &= g^{ns_{i-1}s_i} \cdot (\frac{g^{s_i s_{i+1}}}{g^{s_{i-1}s_i}})^{n-1} \cdot (\frac{g^{s_{i+1}s_{i+2}}}{g^{s_i s_{i+1}}})^{n-2} \cdots \frac{g^{s_{i+n-2}s_{i+n-1}}}{g^{s_{i+n-3}s_{i+n-2}}} \\ &= g^{s_{i-1}s_i + s_i s_{i+1} + s_{i+1}s_{i+2} + \cdots + s_{i+n-2}s_{i+n-1}} \\ &= g^{s_1 s_2 + s_2 s_3 + s_3 s_4 + \cdots + s_n s_1} \end{aligned}$$

In addition to the initialisation phase, the above protocol has three rounds. Each participant needs to sign two messages and verify $2n$ signatures, in addition to the computations involved in performing P.

2.4 A simplified version of the example protocol

We next present a simplified version of the transformed Burmester-Desmedt protocol. In the simplified protocol, U_i $(1 \leq i \leq n)$ performs the following steps:

1. Each user U_i chooses two random numbers $r_i, s_i \in \{0,1\}^k$, computes $Z_i = g^{s_i}$, and then broadcasts $ID_{U_i}||0||r_i||Z_i$.
2. After receiving the broadcast message from all other parties, each user U_i sets $nonce_i = ((ID_{U_1}, r_1), \cdots, (ID_{U_n}, r_n))$ and stores it as state information. U_i computes $X_i = (Z_{i+1}/Z_{i-1})^{s_i}$, $h_i = Z_1||Z_2||\cdots||Z_n$, and $\sigma_{i1} = Sign_{SK_{U_i}}(1||X_i||nonce_i||h_i)$. U_i then broadcasts $ID_{U_i}||1||X_i||\sigma_{i1}$.
3. When U_i receives any message $ID_{U_j}||1||X_j||\sigma_{j1}$ from user U_j $(1 \leq j \leq n, j \neq i)$, it checks that: (1) $U_j \in S$, (2) 1 is the expected sequence number, and (3) $Vrfy_{PK_{U_j}}(1|X_j|nonce_i|h_i), \sigma_{j1}) = 1$. If any of these checks fail, U_i aborts the protocol.

   U_i computes the session key as:

$$\begin{aligned} K_i &= (Z_{i-1})^{ns_i} \cdot (X_i)^{n-1} \cdot (X_{i+1})^{n-2} \cdots X_{i+n-2} \\ &= g^{ns_{i-1}s_i} \cdot (\frac{g^{s_i s_{i+1}}}{g^{s_{i-1}s_i}})^{n-1} \cdot (\frac{g^{s_{i+1}s_{i+2}}}{g^{s_i s_{i+1}}})^{n-2} \cdots \frac{g^{s_{i+n-2}s_{i+n-1}}}{g^{s_{i+n-3}s_{i+n-2}}} \\ &= g^{s_{i-1}s_i + s_i s_{i+1} + s_{i+1}s_{i+2} + \cdots + s_{i+n-2}s_{i+n-1}} \\ &= g^{s_1 s_2 + s_2 s_3 + s_3 s_4 + \cdots + s_n s_1} \end{aligned}$$

It is easy to see that this simplified protocol is more efficient than that obtained from directly applying the compiler in two respects: the protocol has only two rounds, and each user only needs to compute one signature and verify n signatures.

Assuming that the protocol P, i.e., the protocol of Burmester and Desmedt [10], is secure against a passive adversary, it follows that our simplified protocol is also secure against a passive adversary. To assess its security against an active adversary, we first compare our simplified protocol with that created by the Katz-Yung compiler. There are two main differences between these two protocols:

1. The first difference is that in the simplified protocol we combine the first two rounds of the example protocol into one round, and avoid the signature computation.

2. The other difference is that we require each user to compute the signature on both X_i and $Z_1||\cdots||Z_n$ in the second round.

This means that the only difference in the operation of the two protocols is that in the simplified protocol the authentication of messages sent in the first round is conducted in the second round. As a result, it would seem reasonable to conclude that the simplified protocol is as secure as the example protocol given in section 2.3.

3 A new compiler without TTP

Motivated by the analysis of the example protocol in the previous section, we now propose a new compiler that transforms a group key exchange protocol P secure against a passive adversary into an authenticated group key exchange protocol P' with key confirmation which is secure against passive and active adversaries, as well as malicious insiders. We also prove that P' is a secure authenticated key agreement protocol.

3.1 Description of the compiler

We assume that $\Sigma = (\mathit{Gen}, \mathit{Sign}, \mathit{Vrfy})$ is a signature scheme as specified in section 2.1. We also assume that a unique session identifier S_{ID} is securely distributed to the participants before every instance is initiated. Note that in [16] Katz and Shin propose the use of a session identifier to defeat insider attacks.

Suppose a set $S = \{U_i, \cdots, U_n\}$ of users wish to establish a session key, and h is a one-way hash function. Let ID_{U_i} represent U_i's identity for every i $(1 \leq i \leq n)$.

Given P secure against any passive adversary, the compiler constructs a new protocol P', in which each party $U_i \in S$ performs as follows.

1. In addition to all the operations in the initialisation phase of P, each party $U_i \in S$ also generates a verification/signing key pair (PK_{U_i}, SK_{U_i}) by running $Gen(1^k)$.

2. In each round of the protocol P, U_i performs as follows.

   - In the first round of P, each user U_i sets its key exchange history $H_{i,1}$ to be the session identifier S_{ID}, and initialises the round number k to 1. During each round, U_i should synchronise the round number k.
   - When U_i is required to send a message m_i to other users, it broadcasts $M_i = ID_{U_i}||k||m_i$.
   - Once U_i has received all the messages M_j $(1 \leq j \leq n, j \neq i)$, it computes the new key exchange history as:

     $$H_{i,k} = h(H_{i,k-1}||S_{ID}||k||M_1||\cdots||M_n)$$

     Then U_i continues as it would in P upon receiving the messages m_j $(1 \leq j \leq n, j \neq i)$. Note that U_i does not need to retain copies of all received messages.

3. In an additional round, U_i computes and broadcasts the key confirmation message $ID_{U_i}||k||\sigma_i$, where $\sigma_i = Sign_{SK_{U_i}}(ID_{U_i}||H_{i,k}||S_{ID}||k)$.

4. U_i verifies the key confirmation messages from U_j $(1 \leq j \leq n, j \neq i)$. If all the verifications succeed, then U_i computes the session key K_i as specified in protocol P. Otherwise, if any verification fails, then U_i aborts the protocol execution.

In addition to the initialisation phase, the above protocol adds one round to the original protocol and achieves key confirmation. Each participant needs to sign one message and verify n signatures, in addition to the computations involved in performing P. In addition, each participant only needs to store the (hashed) key exchange history. Hence this compiler yields protocols that are more efficient than those produced by the Katz-Yung compiler.

Note that we are using the term key confirmation here in a different sense to that used, for example, in [20]. That is, here key confirmation does not mean that one participant is certain that any other participant has actually computed the session key. Instead it guarantees that, if two or more participants do compute the session key, then it is guaranteed that all copies of this key are identical. This latter meaning is consistent with recent uses of the term. A detailed discussion of key confirmation is given in [7].

3.2 Security analysis

We first briefly introduce our security model, and then prove our security result.

3.2.1 Security model

We consider three kinds of adversary, namely a passive adversary, an active adversary, and a malicious insider. The passive adversary can only eavesdrop on the protocol messages, while the active adversary can completely control the communication network. We do not allow the active adversary to corrupt the long-term key of any honest entity (since we do not discuss forward secrecy), but we do allow the active adversary to corrupt all previous session keys. In our security model we also consider the malicious insiders who manipulate the protocol messages in a manner deviating from the protocol specification. For example, the malicious insiders might try to impersonate another honest participant or mount a different key attack. It should be noted that we do not consider the case when $n-1$ participants are malicious.

Definition 1. A probability $P(k)$ (k is the security parameter) is said to be *negligible* if for every polynomial $f(k)$ there exists an integer N_f such that $P(k) \leq \frac{1}{f(k)}$ for all $k \geq N_f$.

Definition 2. In a protocol instance, a participant *accepts* if it has successfully receives and verifies (if needed) all the messages required by the protocol specification.

Definition 3. An authenticated group key agreement protocol with key confirmation is *secure* if it satisfies the following security requirements:

1. If only a passive adversary is present, any protocol instance successfully ends and all participants compute the same session key[1]. The adversary's advantage over the session key is negligible.

   It is should be noted that the adversary's advantage is defined in the same way as in the protocol P.

2. When an active adversary is present, if one participant accepts in a certain instance, then the probability that any other participant is not involved in this instance is negligible.

3. When an active adversary is present, if all intended participants accept in a certain instance, then the adversary's advantage over the session key held by any participant is negligible.

[1]We assume that no accidental errors such as network failures occur.

4. When an active adversary is present, if all intended participants accept in a certain instance, then the probability that their session keys are different is negligible.

5. In any protocol instance, one or more malicious participants can only succeed with a negligible probability in manipulating the protocol messages in a manner which deviates from the protocol specification.

3.2.2 Security result

Before proving the main theorem, we first prove the following lemma.

Lemma 3.1. *Suppose U_i sends U_j message M_i in round ℓ of an instance of a protocol P with v rounds (where $1 \leq \ell < v$), and suppose also that U_j receives M'_i. Then, if $M'_i \neq M_i$, $H_{i,v} = H_{j,v}$ with a negligible probability, i.e. U_i and U_j will almost certainly possess different key exchange histories in the final round of P.*

Proof. Since h can be regarded as a random oracle, $H_{i,\ell} \neq H_{j,\ell}$, where

$$H_{i,\ell} = h(H_{i,\ell-1}||S_{ID}||\ell||M_1||\cdots||M_i||\cdots||M_n)$$

$$H_{j,\ell} = h(H_{j,\ell-1}||S_{ID}||\ell||M'_1||\cdots||M'_i||\cdots||M'_n)$$

Note that this claim is correct regardless of whether $H_{j,\ell-1} = H_{j,\ell-1}$ and $M_w = M'_w$ for any w $(1 \leq w \leq n, w \neq i)$.

Recursively, it is straightforward to prove that $H_{i,z} \neq H_{j,z}$, for any z $(\ell + 1 \leq z \leq v)$, because $H_{i,z-1} \neq H_{j,z-1}$ and h can be regarded as a random oracle. $\square$

Under the definitions and assumptions in section 3.2.1, we have the following security theorem.

Theorem 3.2. *If h can be considered as a random oracle, the compiler transforms a group key exchange protocol P secure against any passive adversary into an authenticated group key exchange protocol P' with key confirmation which is secure under our definition.*

Proof. Without loss of generality, we assume that P' has v $(v \geq 2)$ rounds.

The first security requirement is straightforwardly satisfied based on the assumption that the original protocol is secure against any passive adversary.

In order to gain a greater advantage than a passive adversary in a certain protocol instance of P', an active adversary must manipulate some of the protocol messages in the first $v - 1$ rounds, because the adversary can gain no advantage if it only manipulates the key confirmation messages in the last round. Without loss of generality, we suppose that the adversary replaces M_i with M_i' which is sent to U_j by U_i in the l-th $(1 \leq l \leq v-1)$ round of the protocol instance. By lemma 3.1, the probability that $H_{i,v} = H_{j,v}$ holds is negligible.

In the v-th round, U_i broadcasts the key confirmation message $ID_{U_i}||k||\sigma_i = Sign_{SK_{U_i}}(ID_{U_i}||H_{i,v}||S_{ID}||k)$, and U_j broadcasts the key confirmation message $ID_{U_j}||k||\sigma_j = Sign_{SK_{U_j}}(ID_{U_j}||H_{j,v}||S_{ID}||k)$, where $H_{i,v} \neq H_{j,v}$ (from above). In order to make any honest participant accept, the active adversary thus needs to block one of these two messages and broadcast a forged key confirmation message. Without loss of generality, we suppose that the adversary wants to forge U_i's key confirmation message, i.e. the adversary wants to forge a signature $\sigma_i = Sign_{SK_{U_i}}(ID_{U_i}||H_{j,v}||S_{ID}||k)$. Because the session identifier S_{ID} is unique, the adversary cannot obtain this signature from U_i in any other protocol instance. Hence, the adversary needs to forge the signature himself, which can only be successfully achieved with a negligible probability based on the assumption that the signature scheme is unforgeable under an adaptive chosen message attack.

Based on these analysis, it is straightforward to verify that security requirements $2-4$ are satisfied.

In order to gain any advantage, one or more malicious participants can manipulate the protocol messages in the first $v-1$ rounds in the following two ways:

1. Replacing the message(s) from one set of honest participants to another set of honest participants.

2. Sending different messages, which must be the same as those required by the protocol, to some honest participants (of which there must be at least two, given our assumption of at most n-2 malicious participants).

In both cases, at least two honest participants will possess different versions of the key exchange history when they generate the key confirmation messages in v-th round. Based on the previous analysis for the active adversary, it is easy to verify that the malicious participant(s) can also only succeed with a negligible probability. As a result, we have proved that P' satisfies the fifth security requirement.

As a result, we have proved that P' is a secure authenticated group key agreement protocol with key confirmation. □

4 A new compiler with TTP

4.1 Description of the compiler

We assume that $\Sigma = (\mathit{Gen}, \mathit{Sign}, \mathit{Vrfy})$ is a signature scheme as specified in section 2.1. We also assume that a unique session identifier S_{ID} is securely distributed to the participants and the TTP before every protocol instance is initiated. In addition, we assume that the TTP acts honestly and is trusted by all the participants.

Suppose a set $S = \{U_i, \cdots, U_n\}$ of users wish to establish a session key, and h is a one-way hash function. Let ID_{U_i} represent U_i's identity for every i ($1 \leq i \leq n$).

Given a protocol P secure against any passive adversary, the compiler constructs a new protocol P', in which each party $U_i \in S$ performs as follows.

1. In addition to all the operations in the initialisation phase of P, the TTP generates a verification/signing key pair $(PK_{\mathrm{TA}}, SK_{\mathrm{TA}})$ by running $\mathit{Gen}(1^k)$, and makes PK_{TA} known to all the potential participants. Each party $U_i \in S$ also generates a key pair (PK_{U_i}, SK_{U_i}) by running $\mathit{Gen}(1^k)$. The TTP is equipped with reliable copies of all the public keys PK_{U_i} ($1 \leq i \leq n$).

2. In each round of the protocol P, U_i performs according to the following rules.

   - In the first round of P, U_i sets its key exchange history $H_{i,1}$ to S_{ID}, and initialises the round number k to 1. During each round, U_i should synchronise the round number k.
   - When U_i is supposed to send message m_i to other users, it broadcasts $M_i = ID_{U_i}||k||m_i$.
   - When U_i receives the message M_j from user U_j ($1 \leq j \leq n$), it computes the new key exchange history as:

     $$H_{i,k} = h(H_{i,k-1}||S_{ID}||k||M_1||\cdots||M_n)$$

     Then U_i continues as it would in P upon receiving the messages M_j. As before, U_i does not need to store copies of received messages.
   - In an additional round, U_i computes and sends the key confirmation message $ID_{U_i}||k||H_{i,t}||\sigma_i$ to the TTP, where

     $$\sigma_i = \mathit{Sign}_{SK_{U_i}}(ID_{U_i}||H_{i,t}||S_{ID}||k)$$

3. The TTP checks whether all the key exchange histories from U_j ($1 \leq j \leq n$) are the same, and verifies each signature. If all these verifications succeed, the TTP computes and broadcasts the signature $\sigma_{\mathrm{TA}} = Sign_{SK_{\mathrm{TA}}}(H_{i,k}||S_{ID}||k)$. Otherwise, the TTP broadcasts a failure message $\sigma_{\mathrm{TA}} = Sign_{SK_{\mathrm{TA}}}(S_{ID}||str)$, where str is a pre-determined string indicating protocol failure.

4. U_i verifies the signature from TA. If the verification succeeds, then U_i computes the session key K_i as required in protocol P. If this check fails, or if U_i receives a failure message from the TTP, then U_i aborts the protocol.

In addition to the initialisation phase, the above protocol adds two rounds to the original protocol and achieves key confirmation. Each participant needs to sign one message and verify one signature, in addition to the computations involved in performing P. In addition, it only needs to store the (hashed) key exchange history. Of course, the TTP needs to verify n signatures and generate one signature.

4.2 Security analysis

In the same security model described in section 3.2.1, we have the following security theorem.

Theorem 4.1. *If h can be considered as a random oracle, the compiler transforms a group key exchange protocol P secure against any passive adversary into an authenticated group key agreement protocol P' with key confirmation which is secure under our definition.*

Proof. Based on the uniqueness of S_{ID} and the security properties (unforgeable under an chosen message attack) of the deployed signature scheme, we make the following two observations regarding a single instance of P':

1. The TTP can verify whether all participants possess the same key exchange history by checking all the signatures σ_i ($1 \leq i \leq n$).

2. Inherently, all honest participants can verify whether all other participants possess the same key exchange history by verifying the signature σ_{TA}. Note that the same key exchange history guarantees that they compute the same session key.

Based on these two observations and the proof of Theorem 3.1, the theorem follows. $\square$

5 Conclusion

In this paper, we have investigated existing methods for building authenticated group key agreement protocols. We have proposed two compilers that transform a group key exchange protocol secure against any passive adversary into an efficient authenticated group key exchange protocol with key confirmation which is secure against any passive adversary, active adversary, or malicious insider. We have shown that protocols generated by both novel compilers are more efficient than those generated by the Katz-Yung compiler.

References

[1] S. Al-Riyami and K. Paterson. Tripartite authenticated key agreement protocols from pairings. In K. G. Paterson, editor, *Proc. IMA Conference on Cryptography and Coding*, volume 2898 of *Lecture Notes in Computer Science*, pages 332–359. Springer-Verlag, 2003.

[2] M. Bellare, R. Canetti, and H. Krawczyk. A modular approach to the design and analysis of authentication and key exchange protocols. In *Proceedings of the thirtieth annual ACM Symposium on the Theory of Computing*, pages 419 – 428. ACM, 1998.

[3] M. Bellare and G. Neven. Transitive signatures based on factoring and RSA. In Y. Zheng, editor, *Advances in Cryptology — Asiacrypt 2002*, volume 2501 of *Lecture Notes in Computer Science*, pages 397–414. Springer, 2002.

[4] M. Bellare, D. Pointcheval, and P. Rogaway. Authenticated key exchange secure against dictionary attacks. In B. Preneel, editor, *Advances in Cryptology — Eurocrypt 2000*, volume 1807 of *Lecture Notes in Computer Science*, pages 139–155. Springer-Verlag, 2000.

[5] M. Bellare and P. Rogaway. Entity authentication and key distribution. In D. R. Stinson, editor, *Advances in Cryptology – Crypto '93*, volume 773 of *Lecture Notes in Computer Science*, pages 110–125. Springer-Verlag, 1993.

[6] M. Bellare and P. Rogaway. Provably secure session key distribution: The three party case. In *Proc. of the 27th ACM Symposium on the Theory of Computing*, pages 57–66. ACM, 1995.

[7] C. Boyd and A. Mathuria. *Protocols for Authentication and Key Establishment*. Springer, 2004.

[8] E. Bresson and D. Catalano. Constant round authenticated group key agreement via distributed computation. In F. Bao, R. Deng, and J. Zhou, editors, *Proc. of PKC 2004*, volume 2947 of *Lecture Notes in Computer Science*, pages 115–129. Springer, 2004.

[9] E. Bresson, O. Chevassut, and D. Pointcheval. Group Diffie-Hellman key exchange secure against dictionary attacks. In Y. Zheng, editor, *Advances in Cryptology — Asiacrypt 2002*, volume 2501 of *Lecture Notes in Computer Science*, pages 497–514. Springer-Verlag, 2002.

[10] M. Burmester and Y. Desmedt. A secure and efficient conference key distribution system. In A. D. Santis, editor, *Advances in Cryptology—EUROCRYPT '94*, volume 950 of *Lecture Notes in Computer Science*, pages 275–286. Springer-Verlag, 1994.

[11] R. Canetti and H. Krawczyk. Analysis of key-exchange protocols and their use for building secure channels. In B. Pfitzmann, editor, *Advances in Cryptology — Eurocrypt 2001*, volume 2045 of *Lecture Notes in Computer Science*, pages 453–474. Springer-Verlag, 2001.

[12] L. Chen and C. Kudla. Identity based authenticated key agreement protocols from pairings. In *Proc. of the 16th IEEE Computer Security Foundations Workshop — CSFW 2003*, pages 219–233. IEEE Computer Society Press, 2003.

[13] K. Y. Choi, J. Y. Hwang, and D. H. Lee. Efficient ID-based group key agreement with bilinear maps. In F. Bao, R. Deng, and J. Y. Zhou, editors, *Proceedings of the 2004 International Workshop on Practice and Theory in Public Key Cryptography (PKC '04)*, volume 2947 of *Lecture Notes in Computer Science*, pages 130–144. Springer-Verlag, 2004.

[14] W. Diffie and M. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, IT-22(6):644–654, 1976.

[15] I. Ingemarsson, D. Tang, and C. Wong. A conference key distribution system. *IEEE Transactions on Information Theory*, 28(5):714–720, 1982.

[16] J. Katz and J. Shin. Modeling insider attacks on group key-exchange protocols. Cryptology ePrint Archive: Report 2005/163, 2005.

[17] J. Katz and M. Yung. Scalable protocols for authenticated group key exchange. In D. Boneh, editor, *Advances in Cryptology — Crypto 2003*, volume 2729 of *Lecture Notes in Computer Science*, pages 110–125. Springer-Verlag, 2003.

[18] Y. Kim, A. Perrig, and G. Tsudik. Communication-Efficient group key agreement. In *Proc. IFIP TC11 16th Annual Working Conference on Information Security*, pages 229–244. Kluwer, 2001.

[19] A. Mayer and M. Yung. Secure protocol transformation via "expansion": from two-party to groups. In *CCS '99: Proceedings of the 6th ACM conference on Computer and Communications Security*, pages 83–92. ACM, 1999.

[20] A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone. *Handbook of Applied Cryptography.* CRC Press, 1997.

[21] D. Steer, L. Strawczynski, W. Diffie, and M. Wiener. A secure audio teleconference system. In H. Krawczyk, editor, *Advances in Cryptology — Crypto '98*, volume 403 of *Lecture Notes in Computer Science*, pages 520–528. Springer-Verlag, 1998.