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

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.

1 Introduction

The case of 2-party authenticated key exchange has been well investigated within the cryptographic community; 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 [1] to the group setting (e.g. [2, 3]). Unfortunately, most of these schemes only assume a passive adversary, so that they are vulnerable to active adversaries who control the communication network. Recently, several provably secure (against both passive and active adversaries) authenticated group key agreement protocols (e.g. [6, 7]) have been proposed.

In this paper we are particularly interested in protocol compilers which transform one kind of protocols into another. In [8], 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 [4] 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. 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 also adds an additional round to the original protocol, but does not achieve key confirmation. 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. Note that proofs of the two main theorems have been omitted for space reasons — these proofs will be given in the full version of the paper.

The rest of this paper is organised as follows. In section 2, we review the Katz-Yung compiler. 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 review the Katz-Yung compiler. With respect to the 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, [9] for a definition). If k is a security parameter, $Gen(1^k)$ generates a pair of public/private keys for the signing and verification algorithms $(Vrfy, Sign)$.

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 is any 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 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 [4] 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 successfully ends. 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 [4], in order to achieve key confirmation a further additional round is required.

3 A new compiler without TTP

In this section we 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 assume that $\Sigma = (Gen, Sign, Vrfy)$ 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. In [5] 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 a protocol P secure against any passive adversary, the new 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 sets 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.

Theorem 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 exchange protocol P' with key confirmation which is secure against any passive adversary, active adversary, or malicious insider.*

4 A new compiler with TTP

We assume that $\Sigma = (Gen, Sign, Vrfy)$ is a signature scheme which is strongly unforgeable under an adaptive chosen message attack. 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 $Gen(1^k)$, and make 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 $Gen(1^k)$. The TTP knows all 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 his key exchange history $H_{i,1}$ to be S_{ID}, and sets 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

     $$\sigma_i = Sign_{SK_{U_i}}(ID_{U_i}||H_{i,t}||S_{ID}||k)$$
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.

Theorem 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 against any passive adversary, active adversary, or malicious insider.*

5 Conclusion

In this paper, we have investigated existing methods for building authenticated group key agreement protocols, and proposed two compilers which can generate more efficient protocols than the Katz-Yung compiler.

References

1. Diffie, W., Hellman, M.: New directions in cryptography. IEEE Transactions on Information Theory **IT-22** (1976) 644–654
2. Burmester, M., Desmedt, Y.: A secure and efficient conference key distribution system. In Santis, A.D., ed.: Advances in Cryptology—EUROCRYPT '94. Volume 950 of Lecture Notes in Computer Science, Springer-Verlag (1994) 275–286
3. Kim, Y., Perrig, A., Tsudik, G.: Communication-efficient group key agreement. In: Proc. IFIP TC11 16th Annual Working Conference on Information Security. (2001) 229–244
4. Katz, J., Yung, M.: Scalable protocols for authenticated group key exchange. In Boneh, D., ed.: Advances in Cryptology — Crypto 2003. Volume 2729 of Lecture Notes in Computer Science, Springer-Verlag (2003) 110–125
5. Katz, J., Shin, J.: Modeling Insider Attacks on Group Key-Exchange Protocols. Cryptology ePrint Archive: Report 2005/163. (2005)
6. Bresson, E., Chevassut, O., Pointcheval, D., Quisquater, J.J.: Provably authenticated group Diffie-Hellman key exchange. In: Proceedings of the 8th ACM Conference on Computer and Communications Security. ACM Press (2001) 255–264
7. Bresson, E., Catalano, D.: Constant Round Authenticated Group Key Agreement via Distributed Computation. In: Proc. of PKC 2004. (2004) 115–129
8. Mayer, A., Yung, M.: Secure protocol transformation via "expansion": from two-party to groups. In: Proceedings of the 6th ACM conference on Computer and communications security. ACM Press (1999) 83–92
9. Bellare, M., Neven, G.: Transitive Signatures Based on Factoring and RSA. In Zheng, Y., ed.: Advances in Cryptology — Asiacrypt 2002. Volume 2501 of Lecture Notes in Computer Science, Springer-Verlag (2002) 397–414