On Key Agreement and Conference Key Agreement

Colin Boyd

Information Security Research Centre
Queensland University of Technology
2 George Street
Brisbane Q4001
AUSTRALIA
email: boyd@fit.qut.edu.au

Abstract. An attack is demonstrated on a previously proposed class of key agreement protocols. Analysis of the attack reveals that a small change in the construction of the protocols is sufficient to prevent the attack. The insight gained allows a generalisation of the class to a new design for conference key agreement protocols.

1 Introduction

Protocols designed for establishing session keys are frequently divided into two types. *Key transfer* protocols rely on one trusted entity to choose the key, which is then transported to the other users involved. By contrast, in *key agreement* protocols each user involved participates in the formation of the key. One possible advantage of key agreement over key transport is that each user can often be sure that the new key is random as long as that user's own input is.

The most well known key agreement protocol is Diffie-Hellman key exchange [5] which exploits the algebraic properties of exponentiation to allow agreement of a shared secret between two users over an insecure channels. It has long been recognised that the Diffie-Hellman protocol is insufficient on its own because an attacker who controls the path between the two users can intercept the messages and masquerade as each of them; as a result the users share different keys with the attacker rather than one with each other. In view of this many different enhancements to the basic Diffie-Hellman protocol have been proposed which provide user authentication in various ways [11].

An examination of the fundamental requirements of key agreement led to a proposed classification of key agreement protocols in a previous paper of this author [3]. This included abstract and concrete protocols divided into three classes. Concrete examples included protocols which do not fall into the Diffie-Hellman paradigm and which may have advantages in efficiency of implementation. In this paper one of these classes is examined in detail. An attack is revealed which shows that the whole class is insecure as described in the earlier paper. Analysis of the attack shows that the weakness results from the incorrect properties of the function used to combine the user inputs, and this allows the protocols to be

simply repaired by re-defining the properties of this function. The insight gained allows the class of protocols to be generalised to a conference key agreement protocol.

2 Key Agreement Protocol Classes

In this section the basic ideas from the previous work [3] are reviewed. A key agreement protocol is defined to involve two users A and B. Each chooses an input, N_A and N_B respectively, randomly from the domain of interest. The conference key K_{AB} will be the result of the operation of some function, f, called the *combining function*, on the two inputs.

$$K_{AB} = f(N_A, N_B)$$

Note that in certain cases f may have additional inputs. The goals of a key agreement protocols are defined as the following.

KA1 Both participants possess K_{AB} which they can verify is new.
KA2 It is infeasible to find K_{AB} by eavesdropping on the protocol, even if the protocol is repeated many times.
KA3 Both participants have equal input into the combining function.
KA4 Both participants know the identity of the other party which may possess K_{AB}.

Property KA2 must hold with the possibility that an attacker knows old values of K_{AB} from previous protocol runs, since this is a normal assumption in protocol analysis. Note that key confirmation is *not* a goal of these properties, so that although A receives assurance that nobody apart from B has the value K_{AB}, she may have no assurance that B did actually get K_{AB}. If such assurance is required extra protocol messages may be added to form a 'handshake' that requires knowledge of K_{AB}.

Since it is necessary only for A and B to make their inputs known to each other, two messages should normally suffice for these protocols. Clearly at least one input to the combining function must remain unknown to an attacker. The classification of protocols previously defined [3] uses this observation to define three classes.

Class 1 Neither N_A nor N_B is kept confidential.
Class 2 Only N_A is kept confidential.
Class 3 Both N_A and N_B are kept confidential.

Class 1 requires that A and B already share a secret and so, although of practical use in some situations, this is of reduced interest. Classes 2 and 3 can work with the use of public key cryptography. (It should be noted that public or shared keys are required in all the extensions of basic Diffie-Hellman.) Class 3 only requires that each of A and B uses the public key of the other to transport its input

confidentially. Class 2 requires that A transports N_A to B using B's public key, and authenticates its origin using A's public key for a digital signature.

The choice of the combining function is critical in achieving the protocol goals. It was proposed that classes 2 and 3 may be implemented using the following bi-one-way function where h is a suitable one-way function. (A function of two inputs is *bi-one-way* if it is one-way when either of the two inputs is fixed.)

$$f(N_A, N_B) = h(N_A) \oplus h(N_B)$$

One advantage of using this function is that nobody, including A or B, can force use of an old key so KA1 will be satisfied. Also its commutativity means that it is reasonable to assert that KA3 is also satisfied. Unfortunately it turns out that in the case of class 2 its properties are not sufficient to guarantee KA2.

3 The Attack

In this section the attack on the protocols in class 2 is described through a particular example. It is true that the attack may not apply if other combining functions are used than in this example, or if extra fields or processing are applied. However the example shows that the argument used to support the protocol security is flawed and so all protocols in this class should be considered dubious.

Consider the following concrete implementation of a protocol in class 2. It is assumed that B's public key K_B is known to A and that A's signature can be verified by B using A's public key. In the following $\{X\}_{K_B}$ denotes encryption with B's public key while $Sig_A(.)$ is A's signature on the message within the brackets.

1. $A \rightarrow B : \{N_A\}_{K_B}, Sig_A(A, B, h(N_A))$
2. $B \rightarrow A : N_B$

The session key is then defined as $K_{AB} = h(N_A) \oplus h(N_B)$. B checks the signature of A against the received value of N_A before accepting the key. As mentioned above, both KA1 and KA3 are satisfied. KA4 is satisfied for A because she knows that only B will receive N_A and is satisfied for B because he can verify that N_A can have been signed only by A. Superficially KA2 appears to hold too because an attacker cannot obtain N_A and consequently cannot obtain K_{AB}. The flaw in this argument is that N_A itself is not actually required to find K_{AB}, but only $h(N_A)$, and this opens the door for a replay attack.

Consider an attacker C who captures the messages passed in an earlier protocol run between A and B. In particular C obtains the value N_B. We may also suppose that C will obtain the session key for K_{AB} for this earlier run. Then C can find $h(N_B)$ and so can find $h(N_A) = K_{AB} \oplus h(N_B)$. C is now able to masquerade as A by simply replaying message 1 from the earlier protocol run. Even though B chooses a new value N'_B, C can still find the new session key $K'_{AB} = h(N_A) \oplus h(N'_B)$.

Observe that the attacker does not learn N_A and does not need to in order to obtain K_{AB}. In terms of the secure channel analysis performed in the original paper there has been no failure of authentication or confidentiality of messages. N_A was indeed conveyed to B in a confidential manner; the problem was simply that N_A was not required in order to find the session key.

4 Fixing the Protocols

It is already made clear that the problem arises because N_A is not required in order to find K_{AB}. The protocol can be fixed by changing only the combining function to one in which it is necessary to know N_A and such that the N_A used cannot be found from knowledge of an old session key. A function with such a property is readily to hand, namely a *message authentication code* (MAC) sometimes called a *keyed one-way hash function*. Recently there has been a great deal of interest in how to construct secure and efficient MACs [1, 2, 9]. However there are a number of alternative definitions of their exact properties so we need to be careful exactly what we require of our function.

A MAC, like a bi-one-way function, is a function of two inputs $f(k, x)$, where the first variable is a secret key. Two basic properties of MACs are that it should not give away its key even after repeated use, and that it is infeasible to calculate $f(k, x)$ without knowledge of k. These are just the property required in the combining function to prevent the replay attack. However, in addition to this property we require that the function also has the properties of a bi-one-way function, namely that it should be one-way in either of the variables when the other is fixed.

1. For a fixed (known) k_0 and output value y_0 it should be infeasible to find a value x with $f(k_0, x) = y_0$.
2. For a fixed x_0 and output value y_0 it should be infeasible to find a k with $f(k, x_0) = y_0$.

Property 1 was included in the definition of keyed hash function given by Berson, Gong and Lomas [2]. However, other authors have pointed out that it is not apparently necessary for many common uses of a MAC [1, 9]. In fact it is reasonable to assert that the common practical methods of MAC construction would probably possess the property, but if more assurance is required then a specific construction is given by Berson *et al.*

Property 2 is a natural property of a MAC as long as the key length is no greater than the output length. This is because if a key could be found which maps x_0 to y_0 then it is likely to be unique and so knowledge of input and output pairs will give away the value of k. The construction of Berson *et al.* does not dictate the size of the parameters, so again may be used as a definite construction if desired.

It is worthwhile to consider carefully once more which properties the protocol has, with the same messages as before, but now using a MAC as the combining function and where N_A takes the role of the MAC key.

KA1 A and B can clearly find K_{AB}. The one-way property in each of the MAC inputs implies that each can verify that the K_{AB} is new as long as they choose their input to be new.

KA2 K_{AB} is now secure from eavesdroppers at least in the attack above. The argument that N_A is never revealed, even with compromise of K_{AB}, and that the N_A value used is required to find any subsequent K_{AB} value now appears sound. Ideally this should be converted to a formal argument but that is beyond the scope of this paper[1].

KA3 One drawback of using the MAC is that the symmetry of the combining function is now lost. However, because the MAC is one-way in both components, neither party can force a particular value and so as far as this goes neither participant controls the session key.

KA4 The argument given above still appears correct. A knows that only B will get N_A, while B can verify that A sent N_A intended for use with B.

5 A New Conference Key Agreement Protocol

One of the advantages of considering the general classes of key agreement is that it gives insight into how to generalise the construction to the conference key situation. With the hindsight of the attack found on the Class 2 protocols with two users, it is clear how they may be generalised. Notice that the Class 1 and Class 3 protocols can also be generalised. For Class 1 though, the situation is even less appealing than with two users, since the assumption that all n users share a secret initially seems unrealistic.

Class 3 protocols could be generalised in an obvious way by having each user encrypt its chosen input value with the public key of each of the other $n-1$ users. These could then be combined by using a *multi-one-way* function which is one-way in each of the n inputs. The obvious generalisation of the bi-one-way function used above would seem adequate here. However, an important disadvantage of this protocol is that each user needs to encrypt and decrypt $n-1$ values which becomes costly for large n. A more efficient option is available for Class 2 and so we concentrate on this for the remainder of this section.

5.1 Proposed Protocol

The generalisation of the Class 2 protocols will involve a set of n users, $\mathcal{U} = \{U_1, U_2, \ldots, U_n\}$. As before each user, U_i, chooses a random value, N_i, in a suitable range. One user, say U_1, will be distinguished and will send its value N_1 to each other user in an authenticated and confidential way. The other users only have to broadcast their messages so that all users in $\mathcal{U}$ receive all the N_i values. U_1 will sign the value N_1 together with the names of all users in the conference. Since this message is the same for every user it only needs to be formed and sent

[1] In general there is no known method for proving a protocol is secure so this is not likely to be a simple task.

once in a broadcast to all users. The value of N_1 is sent to user U_i encrypted with that user's public key, K_i. The protocol then has three stages, the second and third of which each constitute $n-1$ messages. In the following the asterisk is used to denote broadcast messages.

1. $U_1 \rightarrow * : \mathcal{U}, Sig_{U_1}(\mathcal{U}, h(N_1))$
2. $U_1 \rightarrow U_i : \{N_1\}_{K_i}$
3. $U_i \rightarrow * : N_i$

The conference key should then be defined by

$$K_{\mathcal{U}} = f(N_1, h(N_2) \oplus h(N_3) \ldots \oplus h(N_n))$$

where f is a MAC and h is a one-way function. The purpose of h in the definition of $K_{\mathcal{U}}$ will be discussed below. The purpose of h in message 1 is simply to protect the confidentiality of N_1.

Before examining the security of the protocol, let us consider the computational requirements for each user. U_1 has to perform $n-1$ public key encryptions and 1 signature. The other $n-1$ users have only to check one signature and decrypt one message, so for them the computational requirements are the same as for the two user case. U_1 has a high computational burden, and yet this can be reduced substantially by using Rabin's public key cryptosystem [10] for the encryption of N_1. Since encryption with Rabin's scheme only requires one modular multiplication this means that even with a few hundred users, U_1's computational burden is no more than twice that of the other users. (On the other hand it should be acknowledged that use of Rabin's scheme results in a modest increase in computation for the other users.)

5.2 Protocol Security

The arguments for the security of the protocol are much the same as those for the two user case. The key agreement properties should be examined once more.

KA1 All participants evidently possess $K_{\mathcal{U}}$. U_1 derives freshness from the one-way property of the first component of f. Freshness for the other users is derived from use of h and the one-wayness of the second component of f. Even if all other users conspire to choose a specific value, use of h means that they cannot force any specific value for the second component of f, and so cannot force any chosen value for $K_{\mathcal{U}}$.

KA2 Obtaining $K_{\mathcal{U}}$ requires knowledge of N_1 which is available only to users in $\mathcal{U}$. Compromise of old session keys does not reveal N_1 which must be used in order to find the value of any other derived key with N_1 as first component. As for the two user case a formal argument would give greater confidence but appears difficult.

KA3 We would like all participants to have an equal input to $K_{\mathcal{U}}$. As in the two user case the asymmetry of the input of U_1 means there is an obvious difference from other users. However, in the sense that each user cannot

determine the output even with knowledge of the other inputs, they all have equal influence.

KA4 U_1 decides who will obtain N_1, while the other users receive an authenticated message regarding which other users have obtained it.

5.3 Key Confirmation

It is often desired to include in key establishment protocols a method for users to verify that the key they have received has also been received by the other protocol participants. It is worth considering the problem of key confirmation for the conference key protocol, because a simple handshake that might be added to the two-user protocol is not effective and does not appear to be usefully generalised. The problem is that in a two-user protocol each user only needs to know that *some* other entity has the key — if so then that entity must be the second user guaranteed by KA4. In the conference key situation this is of little use because the session key itself cannot be used to distinguish exactly which users have the key.

The simplest solution seems to be to use public keys for each user to sign the conference key and broadcast the signature. Each signature should include all the random challenges and the conference key itself as well as the set of users involved. The values N_1 and $K_{\mathcal{U}}$ need to have their confidentiality maintained and are thus hashed in the following. However this would not be necessary if the signature scheme itself already included a hashing mechanism.

$$U_i \rightarrow * : Sig_{U_i}(h(N_1, K_{\mathcal{U}}), N_2, \ldots, N_n, \mathcal{U})$$

Such a signature needs to be generated and broadcast by each user U_i. The computational effort of this key confirmation phase is now worse than the key agreement protocol itself! Each user must generate one signature and verify $n-1$ others. However, the effort per user can be greatly reduced, this time by using Rabin's signature scheme for which signature verification requires only one modular multiplication. (Another possibility is to use RSA signatures with a small public exponent.)

5.4 Comparison with Previous Protocols

A number of researchers have considered ways to generalise key agreement protocols to include the situation where a group of n users wish to agree a session key. Ingemarsson, Tang and Wong [6] generalised the Diffie-Hellman to a conference, while Burmester and Desmedt developed a more efficient version quite recently [4]. However both of these have dealt only with the confidentiality of the session key and not considered authentication. As a result the man-in-the-middle attack on basic Diffie-Hellman is still applicable to them. A protocol of Klein, Otten and Beth [8] includes conference key distribution as well as detection of cheaters, and as a result is computationally very expensive.

Just and Vaudenay [7] have extended Burmester and Desmedt's protocol to an authenticated version and provide proofs about certain aspects of security. Because their protocols are related to Diffie-Hellman, they obtain the property of *forward secrecy*, that compromise of long-term private keys does not compromise previously used session keys. This is a property that is not obtained in the proposed protocol. However the protocol of Just and Vaudenay is far more computationally expensive than the proposed one, requiring a sub-protocol to be executed between every pair of users.

6 Conclusion

The attack on the previously published class of protocols shows that they are quite insecure with the properties stated. What is quite unusual is that the problem can be fixed without changing the messages sent at all, but rather altering the properties of the combining function used to define the session key.

As is often the case, once an attack is understood it is easy to correct the design, in this case with a minimal change to the original. In addition this greater understanding can lead to insight into other extensions such as the conference protocol proposed above. Readers are invited to consider possible attacks on the new proposed protocols.

Acknowledgements

I am grateful to Anish Mathuria for insightful comments, and to the anonymous referees for pointing out some important omissions.

References

1. S. Bakhtiari, R. Safavi-Naini and J. Pieprzyk, "Keyed Hash Functons, *Cryptography: Policy and Algorithms*, Springer-Verlag, LNCS 1029, pp.210-214, 1996.
2. T. Berson, L. Gong and M. Lomas, "Secure, Keyed and Collisionful Hash Function", Technical Report, SRI International, September 1994.
3. C. Boyd, "Towards a Classification of Key Agreement Protocols", *IEEE Computer Security Foundations Workshop*, pp.38-43, IEEE Press 1995.
4. M. Burmester and Y. Desmedt, "A Secure and Efficient Conference Key Distribution System", *Advances in Cryptology - Eurocrypt 94*, Springer-Verlag, 1995, pp.275-286.
5. W. Diffie and M. Hellman, "New Directions in Cryptography", *IEEE Transactions on Information Theory*, IT-22, 6, pp.644-654, 1976.
6. I. Ingemarsson, D. Tang and C. Wong, "A Conference Key Distribution Scheme", *IEEE Transactions on Information Theory*, IT-28, 5, September 1982, pp.714-720.
7. M. Just and S. Vaudenay, "Authenticated Multi-Party Key Agreement", *Advances in Cryptology - Asiacrypt 96*, Springer-Verlag, 1996, pp.26-35.
8. B. Klein, M. Otten and T. Beth, "Conference Key Distribution Protocols in Distributed Systems", *Codes and Cyphers - Cryptography and Coding IV*, IMA, 1995, pp.225-242.

9. B. Preneel and P. van Oorschot, "MDx-MAC and Building Fast MACs from Hash Functions", *Advances in Cryptology - Crypto '95*, Springer-Verlag, 1995, pp.1-14.
10. M. Rabin, "Digitalized Signatures and Public-Key Functions as Intractable as Factorization", MIT Laboratory for Computer Science, 1979.
11. R. Rueppel and P. van Oorschot, "Modern Key Agreement Techniques", *Computer Communications*, July 1994.

This article was processed using the LaTeX macro package with LLNCS style