Authentication Using Minimally Trusted Servers*

Liqun Chen, Dieter Gollmann and Chris J. Mitchell
Information Security Group, Royal Holloway, University of London,
Egham, Surrey TW20 0EX, England.

{liqun,dieter,cjm}@dcs.rhbnc.ac.uk

Abstract

A number of key distribution protocols using multiple authentication servers, where a minority of them may be untrustworthy, have recently been proposed. This paper analyses the problem of key distribution using minimally trusted multiple servers, and presents a new protocol. In this protocol, as long as all servers do not collude to defraud the clients, either a session key (not known to any server) is successfully established, or the protocol fails in such a way that the clients are aware that it has failed, i.e. the protocol works in a situation where the servers are 'minimally trusted'.

1 Introduction

We suppose that two entities, which do not have appropriate security-related knowledge of one another, want to communicate securely. For this purpose they get assistance from a third party, referred to as an authentication server, which provides them (as clients) with a service including the verification of one another's identity and the establishment of a shared session key. Typically, it is assumed that the server is trustworthy for both clients, and an authentication and key distribution protocol depends on this trust relationship for correctness. However, in some environments where such a protocol is used, the clients have no reason to trust an individual server, e.g. there may be corrupted servers or failed servers, which do not follow the protocol specifications correctly [6]. In such circumstances, the clients cannot rely on the servers to adhere to the protocol rules.

Some recent research [2, 3, 6, 8, 9, 12] has focussed on analysing trust in authentication servers, and constructing secure key distribution protocols which do not require trusting individual authentication servers. A range of possible approaches have been considered.

In one approach, a client can choose which server is trustworthy from a set of authentication servers, typically by applying a security policy or the history of performance and reliability. Yahalom et al. [12] proposed a protocol which allows a client or his agent to choose trustworthy

*This work was jointly funded by the UK EPSRC under grant number GR/J17173, and by the European Commission under ACTS project AC095 (ASPeCT).

servers and avoid untrustworthy ones. One difficulty with this scheme is that a client may sometimes find it difficult to distinguish between trustworthy and untrustworthy servers.

Another approach (see, for example, [4]), based on asymmetric cryptography, is to separate authentication information transfer from authentication information issue, i.e. to let the authentication information issuer be off-line. One instance of this approach is where a master server (sometimes called the authentication information issuer or certification authority) issues a certificate which is then held by another server (sometimes called the authentication information transferrer). The certificate is valid for a period of time, during which there is no need to further contact the master server, since the transferrer can provide the certificate. The client does not need to trust the on-line transferrer, but does need to trust the issuer. This approach does not reduce the need for a single completely trusted server, although this server may now be off-line.

A third approach uses a set of servers simultaneously to achieve authentication. Gong [6] proposed a protocol with multiple authentication servers such that a minority of corrupted and colluding servers cannot compromise security or disrupt service. In that protocol two clients participate in choosing a session key, and each relevant server is responsible for converting and distributing a part of the session key. Two variations on this approach have been described in [3], in both of which the servers each generate a part of a session key, which can be successfully established between a pair of entities as long as more than half the servers are trustworthy. However, in some environments, requiring a majority of the servers to be trustworthy is too 'strong' a condition.

In this paper we consider another variation on the third approach above. We start in Section 2 by discussing possible trust relationships between clients and servers during authentication and key distribution, and in Section 3 we set up our notation and state our fundamental assumptions.

Following this discussion, in Sections 4 and 5 we present two variants of an authentication and key distribution protocol using minimally trusted servers. The new protocol is loosely based on a 3-party authentication and key distribution protocol recently proposed by Steiner et al. [11]. Steiner et al.'s protocol, using Diffie-Hellman key agreement [5], requires the third party to be trustworthy, because a dishonest server can subvert the protocol by impersonating one client to another. However, their protocol has the interesting feature that the resultant session key, although contributed to and 'strengthened' by the server, is not disclosed to him. In the new protocol, clients do not need to trust either an individual server or a majority of the servers. This symmetric cryptography and Diffie-Hellman key agreement based authentication protocol works as long as not all of the servers are untrustworthy and colluding.

We then, in Section 6, briefly consider other possible variants on the basic protocol. The final section contains our conclusions.

2 Trust relationships

Simmons and Meadows [10] pointed out that an information integrity protocol involves a transfer of trust. For example, in a typical 3-party key distribution protocol, the two clients both share secret keys with a server. Each client trusts that the server will protect its key, and authenticate

all client requests and responses. That means both clients believe the server will be honest and competent to follow the protocol specifications correctly. After the protocol succeeds, a session key is established between these two clients. So trust in the server, and in the integrity of the client's secure communication links with the server, has been transferred to trust in the integrity of the communication link, established using the session key, between two clients who had no prior trusted contact. If the server cannot be trusted then neither can the newly established link between clients.

As was mentioned earlier, the two clients may not trust an individual server, and so they might make use of a set of servers to complete identity verification and session key establishment. We now observe what kinds of trust could be transferred. In general, there are four different cases to consider, depending on how many servers are available and how many of them are trustworthy.

1. *Two clients trust a single authentication server.* This is the 'standard' case we have mentioned above.

2. *Two clients trust different authentication servers.* An example is provided by a mobile telecommunications system, where two communicating mobile users, who are registered with different service providers[1], only trust their own service providers. These two service providers collaborate to provide an authentication service to the users.

3. *Two clients trust that at least k authentication servers from a set of n such servers will follow the protocol correctly.* In this case, the clients do not trust any particular servers, but they believe that at least k of the servers, which may be either a majority or a minority, will follow the protocol specifications correctly. One example of this case is two mobile users with a set of network operators[2] in a global mobile telecommunications system. These two users, who are roaming between different networks, get security-related assistance from a set of network operators. Neither user trusts any individual network operator, but they may have reason to believe that some of the network operators are 'good'.

4. *Two clients only believe that a set of servers are not all colluding.* In this case, the clients do not trust any of the servers, but they believe that the servers cannot all collude to defraud them. The example described in the last case still applies, where the mobile users may have reason to believe that the network operators are not all colluding.

The type of trust relationship which may exist between clients and servers depends on the environment in which the protocol is used. When there is no single server trusted by both clients, the clients possibly need to ask two or more untrustworthy servers to provide an authentication service. Such a set of servers, each of whom is of questionable trustworthiness when acting alone, can be believed by the clients to be acceptably trustworthy, when they are compelled to

[1] In a mobile system, a service provider is the body with which a mobile user has a contractual relationship for the provision of telecommunications services.

[2] In a mobile system, a network operator allows users to gain access to the network in order to be able to use the services. A set of network operators may collaborate to provide an authentication service and distribute a session key to two mobile users.

act jointly by an appropriate protocol. After the protocol succeeds, the trust in the joint action of all the servers is transferred to a trust in the security of the communications established using the session key between two clients.

Finally note that no protocol can work correctly if all the servers are untrustworthy and they all collaborate, since they can jointly impersonate one client to another. Hence the minimal possible trust requirement is encapsulated in Case 4 above.

3 Assumptions and notation

The protocols we describe are based on symmetric encipherment and Diffie-Hellman key agreement. It is assumed that two clients A and B want to communicate securely with each other. For this purpose they need to verify each other's identity and establish a shared session key, but before the authentication processing starts they do not share any secret. They therefore make use of third parties, namely a set of servers S_1, ..., S_n. We consider two possible trust relationships, namely cases 3 and 4 discussed above.

We assume that the encipherment operation used provides origin authentication and integrity services, i.e. a received message encrypted using a shared secret must have been sent by the owner of the secret in the form that it was received. We suppose that the Diffie-Hellman key agreement used is based on the Galois Field $\mathbb{Z}_p$ (p prime), and g, a primitive element of $\mathbb{Z}_p^*$. We also suppose that A and B have agreed use of p and g, which may be universally used within some domain of clients, and that p and g are chosen so that the Diffie-Hellman problem is insoluble, i.e. so that computing g^{xy} from knowledge of g, p, g^x, and g^y (for any $x, y \in \mathbb{Z}_p^*$) is computationally infeasible.

In the protocol descriptions, we make use of the following notation.

- $\{X\}_K$ is the result of the encipherment of data X with a symmetric encipherment algorithm using the key K.
- h is a collision-free, one-way hash function.
- $D_Q \in N$ is entity Q's private key agreement value and R_Q ($= g^{D_Q} \bmod p$) is Q's public key agreement value. R_Q and D_Q are changed in every use of the protocol.
- X, Y is the result of the concatenation of data items X and Y.
- $P \rightarrow Q : Z$ denotes that P sends message Z to Q.
- K_{AB} ($= h(g^{D_A D_B} \bmod p)$) is the session key shared by A and B as a result of the protocol.

Finally observe that we abuse our notation slightly and use A and B to indicate both entities and identifiers for these entities. The exact meaning should be clear from the context.

4 The basic protocol

We first describe a simple version of our protocol, which operates in the 'minimal trust' case, where no servers are trusted to behave correctly. We only assume that not all the servers will collude together. This protocol thus applies to any of the trust relationships described above, in particular it applies even in Case 4. We actually elaborate this trust case slightly. For the purposes of this protocol suppose that A and B believe that no more than $n-k$ of the n servers can collude to defraud them (the case $k = 1$ corresponds to the situation described in Case 4 above). In fact the protocol could be modified in a trivial way to allow A and B to choose different values of k, but, for the sake of simplicity we assume they choose the same value.

However, as we will see, if a secret value used by one of the legitimate parties in any instance of the protocol is compromised, then the protocol is vulnerable to attack. In a subsequent section we describe a more elaborate protocol which offers of measure of protection against attacks based on a compromise of 'old secrets'.

4.1 Protocol description

The protocol has five 'rounds' of exchanged messages: $M_1, M_{2i}, M_{3i}, M_4, M_5$, where messages M_{2i} and M_{3i} are sent for every i $(1 \leq i \leq n)$. If, at any point in the protocol, A or B decide to reject one of the servers S_i, in all subsequent messages the information corresponding to S_i is replaced by an appropriate error message.

A initially chooses and stores D_A secretly, calculates $R_A = g^{D_A} \bmod p$, enciphers n blocks made up of R_A and B using K_{AS_i} $(1 \leq i \leq n)$, and sends these n blocks together with its own name to B in M_1.

$$M_1: \quad A \rightarrow B: \quad A, \{R_A, B\}_{K_{AS_1}}, \{R_A, B\}_{K_{AS_2}}, \ldots, \{R_A, B\}_{K_{AS_n}}$$

On receiving M_1, B chooses and stores D_B secretly, calculates $R_B = g^{D_B} \bmod p$, enciphers n blocks made up of R_B and A using K_{BS_i} $(1 \leq i \leq n)$, and sends the ith enciphered block with A's ith enciphered block and both clients' names to S_i in M_{2i} $(1 \leq i \leq n)$.

$$M_{2i}: \quad B \rightarrow S_i: \quad A, B, \{R_A, B\}_{K_{AS_i}}, \{R_B, A\}_{K_{BS_i}}$$

On receipt of M_{2i}, S_i deciphers both parts, enciphers two blocks made up of R_A, R_B and A or B using K_{BS_i} or K_{AS_i} respectively, and then sends them in M_{3i} $(1 \leq i \leq n)$ to B.

$$M_{3i}: \quad S_i \rightarrow B: \quad \{R_A, R_B, B\}_{K_{AS_i}}, \{R_B, R_A, A\}_{K_{BS_i}}$$

After receiving M_{3i}, B deciphers its second part using K_{BS_i}, $1 \leq i \leq n$. B next checks whether the decrypted half of each message M_{3i} contains both the correct value of R_B and the expected identifier for A. If not, B rejects the server S_i. Similarly, if no response is obtained from S_i before some predefined time-out, B will again reject this server. B now compares the values of R_A in the decrypted second half of all remaining messages M_{3i} (i.e. all those not so far rejected). If at least $n-k+1$ of the values agree on a value R_A, then B provisionally accepts this value of R_A, and rejects any servers supplying a different value of R_A. If there are two or more sets of

$n-k+1$ messages, each agreeing on a different value of R_A, B selects one of the values R_A by some strategy, e.g. at random or using B's preferred servers (if he/she has any), and rejects all messages not providing this value; of course, this is unlikely to arise in practice, since it could only arise by active interference with messages M_{2i} sent to a number of servers.

Of course, if k or more servers have already been rejected prior to this point then there is no chance of proceeding. As soon as such an event arises B must either restart the protocol and send messages M_{2i} again, or ask A to restart the protocol from message M_1.

B now computes its provisional value of K_{AB} as $K = (R_A)^{D_B}$, then calculates $h(K, A)$, and sends M_4 to A.

$$M_4: \quad B \rightarrow A: \quad \{R_A, R_B, B\}_{K_{AS_1}}, \{R_A, R_B, B\}_{K_{AS_2}}, \ldots, \{R_A, R_B, B\}_{K_{AS_n}}, h(K, A),$$

if necessary replacing that part of the message corresponding to a rejected server with an error message.

After receiving message M_4, A deciphers the n enciphered blocks using K_{AS_i} $(1 \leq i \leq n)$ and checks them in a similar way to B. A first rejects any message with an incorrect value of R_A or an incorrect identifier B. As for B, if at any point A finds that k or more servers have been rejected then A must restart the protocol by sending M_1 again. A now compares the values of R_B in the decrypted text of all remaining blocks (i.e. all those not so far rejected). If at least $n-k+1$ of the values agree on a value R_B, then A provisionally accepts this value of R_B.

A now computes its provisional value of K_{AB} as $K' = {R_B}^{D_A}$, next calculates $h(K', A)$, and then compares this hash-value with the hash-value received from B. If it matches, A has confirmation that the provisionally accepted value of R_B is correct (and hence $K_{AB} = K'$ is correct), and that B has the key K_{AB}; A has also authenticated B. A now computes $h(K', B)$ and sends message M_5 to B.

$$M_5: \quad A \rightarrow B: \quad h(K', B)$$

On receipt of M_5, B computes $h(K, B)$ and compares it with the value in M_5. This confirms that the provisionally accepted value of R_A is correct (and hence $K_{AB} = K$ is correct), and that A has obtained K_{AB}; B has also authenticated A.

4.2 Security analysis

Theorem 1 *If the protocol succeeds, then A and B will have:*

- *mutually authenticated one another,*
- *agreed on the same key K_{AB}, which will equal $h(g^{D_A D_B})$, where D_A and D_B are the values chosen by A and B during the protocol, and which will not be known to any server,*
- *provided mutual key confirmation.*

Proof: To establish this result we proceed in a step-wise way. We first consider the value of R_A accepted by B after receipt of messages M_{3i}. We wish to establish that this is a value

of R_A chosen by A at some point in the past (either in this 'run' of the mechanism or in some previous run). First observe that we assumed that the encryption mechanism in use provides origin authentication and data integrity, and thus successful decipherment of a message implies that the recipient has guarantees that the content came from an entity which knows the relevant secret. Thus each server S_i, after receiving message M_{2i}, will either obtain values of R_A and R_B that were 'correct' at some point in the past, or will know that the message has been interfered with.

Now, since we assume that at most $n-k$ servers can collude and B will only provisionally accept R_A if the same value if provided by $n-k+1$ servers, then we know that a value R_A provisionally accepted by B (after receipt of M_{3i}) must correspond to a value chosen by A at some time in the past, and we have our desired result.

We briefly observe that it is theoretically possible for two or more values of R_A to be acceptable, i.e. for $n-k+1$ of the messages to agree on one value, another $n-k+1$ messages to agree on a different value, and so on. This could happen (given k is large enough) if a sufficiently large number of messages M_{2i} were replaced with corresponding messages from a previous iteration of the protocol. As stated in the description of the algorithm, B's strategy in such an event is to accept one of the values, and if the wrong one is accepted then the protocol will fail.

By a similar argument we can show that the value of R_B provisionally accepted by A must be a value of R_B selected by B for use with A at some point in the past.

We now consider what we can conclude if A finds that the hash value in message M_4 matches the hash-code $h(K', A)$ computed using K' (A's value of K_{AB}). Since we assumed that h is collision-free, this implies that $K = K'$. Now, A knows that $K' = g^{D_A D_B}$ for some value of D_B chosen at some point in the past by B (and known only to B). Hence, since $K = K'$, A knows that either K must have been computed as $K = R_A^{D_B}$ or else some third party, knowing only $R_A = g^{D_A}$ and $R_B = g^{D_B}$, has found a way to compute $g^{D_A D_B}$. But this is impossible by our assumption regarding the difficulty of the Diffie-Hellman problem. But only B can have computed $K = R_A^{D_B}$, since only B knows D_B. Hence R_B is the current 'fresh' value chosen by B, since B has used it with R_A, and A has confirmation that K_{AB} is the 'correct' shared key and is known by B (key confirmation). A has also authenticated B. Note that it was necessary for our reasoning that no value of D_B ever used by B in this protocol has become compromised.

Exactly similar arguments apply to the processing of message M_5 by B (again noting that it is necessary that no value of D_A ever chosen by A has become compromised. The result now follows. □

Remark 1 *As noted above, the protocol we have just described requires all the values D_A and D_B which have ever been used by A and B to remain uncompromised. In practice this is not an unreasonable assumption. However, for completeness, in the next section we describe a protocol which remains secure even if such values become compromised (subject to certain assumptions).*

Of course, it should be clear that any group of k malicious servers can disrupt the protocol by sending incorrect values in messages M_{3i}. Moreover any entity can prevent the protocol succeeding by manipulating and/or suppressing some of the exchanged messages. This second point will, of course, be just as true for any protocol. The first point is potentially a little more serious, since we are dealing with 'untrustworthy' servers.

However, in practice, authentication servers (e.g. network operators or service providers in mobile telecommunications networks) will normally be expected to operate correctly. The purpose of a protocol of the type we have just described is to detect untrustworthy behaviour in the unlikely event that it occurs, not to correct persistent faulty behaviour. If one or more servers consistently fails to follow protocol specifications correctly, then we would expect them to be removed from the network (or at least gain a very bad reputation!).

5 A more robust protocol

We now consider a variant of our protocol which again operates in the 'minimal trust' case, where no servers are trusted to behave correctly, and which also remains secure even if previously used secret values D_A or D_B become known to a server.

As in the previous section we suppose that A and B believe that no more than $n-k$ of the n servers can collude to defraud them (the case $k=1$ corresponds to the situation described in Case 4 above).

5.1 Protocol description

The first two message exchanges (i.e. M_1 and M_{2i}, $1 \leq i \leq n$) are exactly as for the previous case, as is the associated checking.

On receipt of M_{2i}, server S_i $(1 \leq i \leq n)$ chooses a random number T_i and sends message M_{3i} $(1 \leq i \leq n)$ to B.

$$M_{3i}: \quad S_i \rightarrow B: \quad \{R_A, R_B, B, T_i\}_{K_{AS_i}}, \{R_B, R_A, A, T_i\}_{K_{BS_i}}$$

After receiving M_{3i}, $1 \leq i \leq n$, B deciphers the second part of each message using K_{BS_i}. B next checks that the decrypted half of each message M_{3i} contains both the correct value of R_B and the expected identifier for A; if not, B rejects the server S_i. As in the previous protocol, if no response is obtained from S_i before some predefined time-out, B will reject this server. B now obtains the values of R_A from the decrypted second half of all received accepted messages M_{3i} (suppose there are m different such values $(1 \leq m \leq n)$), and uses these values in conjunction with D_B to compute a series of 'candidate' values for K_{AB}: label these values $K_1, K_2, \ldots, K_m$.

As previously, if k or more servers have already been rejected prior to this point then there is no chance of proceeding. As soon as such an event arises B must either restart the protocol and send messages M_{2i} again, or ask A to restart the protocol from message M_1.

B now enumerates every $(n-k+1)$-subset of $\{1, 2, \ldots, n\}$ (i.e. all $\binom{n}{k-1}$ of them), and for each such subset $W = \{w_1, w_2, \ldots, w_{n-k+1}\}$, computes the m values:

$$\begin{aligned} &h(K_1, T_{w_1}, T_{w_2}, \ldots, T_{w_{n-k+1}}, A), \\ &h(K_2, T_{w_1}, T_{w_2}, \ldots, T_{w_{n-k+1}}, A), \ldots, \\ &h(K_m, T_{w_1}, T_{w_2}, \ldots, T_{w_{n-k+1}}, A) \end{aligned}$$

using the values of $T_1, T_2, \ldots, T_n$ recovered from the messages M_{3i}. Note that, when concatenated together as input to the hash-function, the values T_{w_i} are always assembled in an agreed

order (e.g. alphabetical/numerical order of server IDs).

B now sends M_4 to A, containing the first half of the n messages from the servers S_i, and all $m\binom{n}{k-1}$ computed values:

$$\begin{aligned} M_4: \quad B \rightarrow A: \quad & \{R_A, R_B, B, T_1\}_{K_{AS_1}}, \{R_A, R_B, B, T_2\}_{K_{AS_2}}, \ldots, \{R_A, R_B, B, T_n\}_{K_{AS_n}}, \\ & h(K_1, T_{w_1}, T_{w_2}, \ldots, T_{w_{n-k+1}}, A), \\ & h(K_2, T_{w_1}, T_{w_2}, \ldots, T_{w_{n-k+1}}, A), \ldots \\ & h(K_m, T_{w_1}, T_{w_2}, \ldots, T_{w_{n-k+1}}, A), \ldots \end{aligned}$$

where $W = \{w_1, w_2, \ldots, w_{n-k+1}\}$ ranges over all $(n-k+1)$-subsets.

Note that, if necessary, B replaces that block of the first n blocks of the message corresponding to a rejected server with an error message.

After receiving message M_4, A deciphers the first n blocks using K_{AS_i} $(1 \leq i \leq n)$; A rejects any block if the identifier or the value R_A are not the expected values. A now collects the various different values of R_B that the accepted blocks contain (suppose there are m' different such values $(1 \leq m' \leq n)$), and uses these values in conjunction with D_A to compute a series of 'candidate' values for K_{AB}: label these values $K'_1, K'_2, \ldots, K'_{m'}$.

A now enumerates every $(n-k+1)$-subset of $\{1, 2, \ldots, n\}$ (i.e. all $\binom{n}{k-1}$ of them), and for each such subset $W = \{w_1, w_2, \ldots, w_{n-k+1}\}$, computes the m' values:

$$\begin{aligned} & h(K'_1, T_{w_1}, T_{w_2}, \ldots, T_{w_{n-k+1}}, A), \\ & h(K'_2, T_{w_1}, T_{w_2}, \ldots, T_{w_{n-k+1}}, A), \ldots, \\ & h(K'_{m'}, T_{w_1}, T_{w_2}, \ldots, T_{w_{n-k+1}}, A) \end{aligned}$$

using the values of $T_1, T_2, \ldots, T_n$ recovered from the first n blocks of message M_4.

A now pairwise compares all the newly computed values (all $m'\binom{n}{k-1}$ of them) with the $m\binom{n}{k-1}$ values in the second half of message M_4. If any agreement is found, then A immediately adopts as the shared key K_{AB} the value of K'_i used to compute its matching value.

A then computes $h(K_{AB}, T_{w'_1}, T_{w'_2}, \ldots, T_{w'_{n-k+1}}, B)$ for precisely that $(n-k+1)$-subset $W' = \{w'_1, w'_2, \ldots, w_{n'-k+1}\}$ of $\{1, 2, \ldots, n\}$ for which a match was found during the processing of message M_4, and sends message M_5 to B. Otherwise, A sends B an error message.

$$M_5: \quad A \rightarrow B: \quad h(K_{AB}, T_{w'_1}, T_{w'_2}, \ldots, T_{w'_{n-k+1}}, B)$$

On receipt of M_5, B again enumerates every $(n-k+1)$-subset of $\{1, 2, \ldots, n\}$ (i.e. all $\binom{n}{k-1}$ of them), and for each such subset $W = \{w_1, w_2, \ldots, w_{n-k+1}\}$, computes the m values:

$$\begin{aligned} & h(K_1, T_{w_1}, T_{w_2}, \ldots, T_{w_{n-k+1}}, B), \\ & h(K_2, T_{w_1}, T_{w_2}, \ldots, T_{w_{n-k+1}}, B), \ldots, \\ & h(K_m, T_{w_1}, T_{w_2}, \ldots, T_{w_{n-k+1}}, B) \end{aligned}$$

using the values of $T_1, T_2, \ldots, T_n$ recovered from the messages M_{3i}.

B now pairwise compares all the newly computed values (all $m\binom{n}{k-1}$ of them) with the hash value in message M_5. If any agreement is found, then B immediately adopts that particular value of K_i as the shared key K_{AB}.

5.2 Security analysis

Theorem 2 *If the protocol succeeds, then A and B will have:*

- *mutually authenticated one another, and*
- *agreed on the same key K_{AB}, which will equal $h(g^{D_A D_B})$, where D_A and D_B are the values chosen by A and B during the protocol, and which will not be known to any server,*

and B will have 'key confirmation' (i.e. B will have confirmation that A has the shared key K_{AB}), although A will not have such confirmation until a message is received from B secured using the shared key.

Proof: First consider how a 'match' might occur in the comparisons that A performs after receipt of message M_4. Suppose that

$$h(K_i, T_{w_1}, T_{w_2}, \ldots, T_{w_{n-k+1}}, A) = h(K'_j, T_{w'_1}, T_{w'_2}, \ldots, T_{w'_{n-k+1}}, A),$$

for some i, j and $W = \{w_1, w_2, \ldots, w_{n-k+1}\}$.

Given that h is collision-free (as in our assumptions), it must hold that

$$K_i = K'_j, \quad \text{and} \quad w_s = w'_s \ (1 \leq s \leq n-k+1).$$

Now, since no more than $n-k$ of the servers can collude (by our trust assumption), it follows that no single server knows all the values $T_{w_1}, T_{w_2}, \ldots, T_{w_{n-k+1}}$. Moreover, because A checks the identifier of B in all the server-generated blocks it receives, A also knows that servers $S_{w_1}, S_{w_2}, \ldots, S_{w_{n-k+1}}$ all claim to have sent the respective values $T_{w_1}, T_{w_2}, \ldots, T_{w_{n-k+1}}$ to B. Thus, on the assumption that no more than $n-k$ servers collude, i.e. no more than $n-k$ servers can agree to tell the same lie to A, only B could possibly know all the values necessary to compute the message, and hence B must have sent the value:

$$h(K_i, T_{w_1}, T_{w_2}, \ldots, T_{w_{n-k+1}}, A).$$

Moreover, the message must be 'fresh', since all the values of T_{w_i} were sent with the correct value of R_A.

Now since A computed K'_j, it is clear that $K'_j = g^{D_A x}$ for some value x, where g^x was sent to A by one of the servers. Similarly, since we know that B computed K_i, and since we know it is a fresh message, it is clear that $K_i = g^{D_B y}$ for some value y, where g^y was sent to B by one of the servers, and where D_B is B's 'current' secret.

We know that

$$g^{D_A x} = g^{D_B y}$$

i.e. either $x = D_B$ and $y = D_A$ (which is what we desire to show) or a server (or pair of servers) have found a value $z = xy^{-1}$, where y^{-1} is computed modulo $p-1$, such that

$$(g^{D_A})^z = g^{D_B}$$

without knowledge of D_A or D_B. But this is impossible by our assumption regarding the difficulty of the Diffie-Hellman problem. The fact that a previous value of D_B may have been compromised does not pose a threat in this case.

Hence A knows that K_j' was computed using the correct (fresh) value of D_B, and thus A has accepted the 'correct key'. Thus we have established that:

- A can only accept the correct key K_{AB}, and
- A has authenticated B.

However A does not have confirmation that B knows which is the correct key, since B only has the correct key amongst a set of candidate keys.

By an exactly analogous argument, we can deduce that B will:

- only accept the correct key K_{AB}, and
- authenticate A.

In addition, and unlike the situation for A, B will have confirmation that A knows the correct key. □

Remark 2 *The definition of servers 'not colluding' is implicit in the above proof. We assumed that $n-k+1$ values of T_i cannot be known by anyone apart from B. Of course, if the servers deliberately reveal their values of T_i, then the protocol may fail. However, a server gains little by revealing secrets (unless they are part of a conspiracy), and hence such a possibility does not seem to be a serious practical threat.*

As for the protocol described in Section 4, any group of k servers can prevent A and B from establishing a key by sending different values of T_i to A and B. However, again as before, all they can do is ensure that the protocol fails, and they cannot force A or B to accept anything other than the 'correct' key.

Further if one or both of the keys K_{AS_i} and K_{BS_i} are compromised for t of the servers S_i, by which we mean that the keys are publicly known to all malicious parties (but A and B are not aware of the compromise), then the protocol will still work correctly as long as no more than $n-k-t$ servers collude.

6 Other variants of the protocol

6.1 Relaxed trust requirements

Observe that the two variants of the protocol described in Sections 4 and 5 work in Case 4 of the possible trust relationships defined in Section 2. The processing of the protocol can potentially be relaxed if A and B can trust the set of servers to a greater extent, e.g. as in Case 2. To see this we give an example.

Suppose A and B trust different but identified servers, e.g. their own service providers in a mobile telecommunications system. Note that A (and B) only knows which server it trusts, and does not know which one is trusted by the other party. This is why they might still make use of n servers ($n > 2$), not just the two servers trusted by one of them. A and B now only need to check whether or not a particular message from their own trustworthy server has been received; if it has, then any other message with a different R_B or R_A can be rejected and further processing will continue.

6.2 K_{AB} 'strengthened' by servers

There may be a need for the session key K_{AB} to be 'strengthened' by servers. We give an example of a possible reason for this. In the protocol variants described previously, if any symmetric key K_{AS_i} shared between A and S_i is known to B, B could obtain R_A from M_1 and then control the resultant session key by selecting D_B. In order to prevent this, the servers may be asked to provide a contribution to the session key.

In such a case, exactly the same message exchanges as employed in Section 5 can be used, except that the final shared key is

$$K'_{AB} = g^{D_A D_B T_{w_1} T_{w_2} \ldots T_{w_{n-k+1}}}$$

where $\{T_{w_1}, T_{w_2}, \ldots, T_{w_{n-k+1}}\}$ is the set of values used to compute message M_5.

7 Conclusions

In this paper we have analysed the possible trust relationships between clients and servers during authentication and key distribution, and proposed a protocol using minimally trusted servers. The protocol is based on symmetric cryptography and Diffie-Hellman key agreement. On the assumption that not all servers are untrustworthy and colluding, no untrustworthy server can subvert the protocol either by impersonating one client to another or compromising the security of communications between two clients. Even if the symmetric keys shared by clients and servers are compromised, any malicious third party (e.g. an interceptor) cannot discover the resultant session key, if it is agreed by two clients.

The protocol does not meet likely legal requirements for warranted interception [7], because only the two users know the resultant session key if the protocol is successful. In some cases, governments have requirements to intercept user traffic in order to combat crime and protect national security, see e.g. [1]. So, a possible topic for future research in this area is how to provide an authentication and key distribution service in which no untrustworthy server can compromise the users' secrets, as well as meeting the legal requirements for warranted interception in the domains it serves.

References

[1] National Institute of Standards and Technology. FIPS Publication 185: Escrowed Encryption Standard. February 1994.

[2] E.F. Brickell and D.R. Stinson. Authentication codes with multiple arbiters. In C. G. Günther, editor, *Lecture Notes in Computer Science 330, Advances in Cryptology: Proc. Eurocrypt '88*, pages 51–54. Berlin: Springer-Verlag, 1988.

[3] L. Chen, D. Gollmann, and C.J. Mitchell. Distributing trust amongst multiple authentication servers. *Journal of Computer Security*, 3:255–267, 1994/1995.

[4] Comité Consultatif International de Télégraphique et Téléphonique. CCITT Recommendation X.509 - 1988, The directory - Authentication framework. 1988.

[5] W. Diffie and M.E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22:644–654, November 1976.

[6] L. Gong. Increasing availability and security of an authentication service. *IEEE Journal on Selected Areas in Communications*, 11:657–662, 1993.

[7] N. Jefferies, C. Mitchell, and M. Walker. A proposed architecture for trusted third party services. In E. Dawson and J. Golić, editors, *Lecture Notes in Computer Science 1029, Cryptography: Policy and Algorithms Conference*, pages 98–104. Springer-Verlag, 1996.

[8] B. Klein, M. Otten, and T. Beth. Conference key distribution protocols in distributed systems. In P. G. Farrell, editor, *Codes and Cyphers, Proceedings of the Fourth IMA Conference on Cryptography and Coding*, pages 225–241. Formara Limited. Southend-on-sea. Essex, 1995.

[9] T.P. Pedersen. A threshold cryptosystem without a trusted party. In D. W. Davies, editor, *Lecture Notes in Computer Science 547, Advances in Cryptology: Proc. Eurocrypt '91*, pages 522–526. Berlin: Springer-Verlag, 1991.

[10] G. J. Simmons and C. Meadows. The role of trust in information integrity protocols. *Journal of Computer Security*, 3:71–84, 1994/1995.

[11] M. Steiner, G. Tsudik, and M. Waidner. Refinement and extension of encrypted key exchange. *ACM Operating Systems Review*, 29(3):22–30, 1995.

[12] R. Yahalom, B. Klein, and T. Beth. Trust-based navigation in distributed systems. European Institute for System Security, Karlsruhe University, Technical Report 93/4, 1993.