Journal of Computer Security 3 (1994/1995), 255–267
IOS Press

DISTRIBUTING TRUST AMONGST MULTIPLE AUTHENTICATION SERVERS[1,2]

Liqun Chen, Dieter Gollmann and Christopher J. Mitchell
Information Security Group
Royal Holloway, University of London
Egham, Surrey TW20 0EX, UK
e-mail: {liqun,dieter,cjm}@dcs.rhbnc.ac.uk

Abstract
Some recent research on key distribution systems has focussed on analysing trust in authentication servers, and constructing key distribution protocols which operate using a number of authentication servers, a minority of them may be untrusted. This paper proposes a key distribution protocol with multiple authentication servers using a cross checksum scheme. In the protocol, multiple untrusted servers generate candidate session keys for two clients. The clients use the cross checksum scheme for the verification of these candidate keys. It is not necessary for the clients to trust an individual server. A minority of corrupted and colluding servers cannot compromise security, and their inappropriate behaviour can be detected. Comparing our protocol to similar proposals, we will consider the number of messages exchanged, the length of messages, and the method of key generation.

1. Introduction

In the context of symmetric cryptography, if two entities sharing no secret want to securely communicate with each other, they typically do so with the assistance of a third party. This third party is an authentication server which provides an authentication service including distributing a secure session key to these entities as clients. Such an authentication server is sometimes referred to as a trusted third party since every client has to trust it by sharing a secret with it. The security of a typical key distribution protocol depends on the assumption that the authentication server is trustworthy. If the authentication server is corrupted, or is compromised, the security of communications between these clients cannot be guaranteed.

[1] This work is a part of the DTI/EPSRC Link Personal Communication Programme project 'Security Studies for Third Generation Mobile Telecommunications Systems' and is funded by the UK EPSRC under research grant GR/J17173.

[2] This paper is a revised version of a presentation given at the 8th IEEE Computer Security Foundations Workshop, Dromquinna Manor, Ireland, June 13–15, 1995.

In order to make a protocol work in an environment where clients do not want to trust an individual server, it is worthwhile to find robust authentication schemes which can tolerate corrupted servers. Some recent research [BrSt'88], [Go'93], [KOB'95], [Pe'91], [YKB'93] has focussed on analysing trust in authentication servers, and constructing secure key distribution protocols which do not require trusting individual servers. A range of possible approaches have been considered in which the use of a more complex authentication service results in an authentication service with higher security and availability.

One approach is to allow a client to choose which server is trustworthy and which is untrustworthy from a set of authentication servers, typically by applying a security policy or the history of performance and reliability. Yahalom et al. [YKB'93] proposed a protocol which allows a client or his agent to choose trustworthy authentication 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, and one which forms the basis of the schemes in this paper, uses many servers simultaneously to achieve authentication. In this case, clients do not trust any individual server, but they believe that some of the servers will follow the protocol specifications correctly. One example of this case is two mobile telecommunications users with a set of network operators [3] in a global mobile telecommunications system. These two mobile users, who are roaming between different networks, could get security-related assistance from a set of network operators. Neither user necessarily trusts any individual network operator, but they may have reason to believe that some of the network operators are 'good'.

Gong [Go'89] 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 one client chooses a session key, and each server transfers a part of the key to the other client. In order to prevent one client from imposing a session key on the other, Gong [Go'93] later proposed another protocol, in which two clients participate in choosing a session key, and each server is responsible for converting and distributing a part of the key.

For the following two reasons it makes sense that we consider a new protocol in this paper.

- ◇ In some environments, it is necessary to let servers, not clients, choose a session key, for example, where clients cannot randomly and securely generate a session key or candidate keys. The clients could be capable of generating nonces but are not authorised to generate session keys, because session keys might need to be more random and secure than nonces.
- ◇ It is attractive to find a new protocol that does not require a secret sharing scheme for constructing the session key.

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

One potential problem with letting servers be involved in choosing candidate session keys is in providing clients with the means to verify that they are provided with 'good' candidate keys. In this paper, a candidate session key is 'good' if the same value is received by both clients. In order to solve this problem, we discuss a cross checksum scheme, which works on the assumption that more than half the servers follow the protocol specifications correctly. We let all servers participate in choosing candidate session keys. Each server randomly and securely generates a candidate key. Two clients, wishing to check which candidate keys have been correctly received by both of them, verify this by exchanging the relevant check values using a globally known one-way hash function [DiHe'76], [Me'90]. The clients then eliminate the 'bad' candidate keys and use the 'good' keys to compute a final session key. It is not necessary for the clients to trust any individual server because no single server can know the session key.

Based on this scheme we propose a key distribution protocol with an arbitrary number of authentication servers. In this protocol a minority of corrupted and colluding authentication servers cannot compromise security, and their inappropriate behaviour can be detected. The protocol is arranged in a 'parallel' version and a 'cascade' version, depending on the flows of exchanged messages. If more than half the servers follow the protocol specifications correctly, then the following properties will hold for the session key:

◇ it will be fresh (i.e. not a replay of an old key) and unpredictable (i.e. not predictable by any party),

◇ it will be known only to the clients and not to any server,

◇ it can be checked by both clients, and

◇ it will not have been chosen by any individual client or server.

The two versions of the protocol with n servers we discuss in this paper both possess these four properties, and use $2n+4$ and $n+5$ messages respectively, whereas Gong's protocol [Go'93] which also has these properties uses $4n+3$ messages. Gong's other protocol in [Go'89] uses only $2n+2$ messages, but that protocol lets one client choose a session key, so that the last property does not hold.

The rest of the paper is organised as follows. The relevant notation and assumptions made in this paper are described in the next section. A typical authentication protocol and how trust problems can arise are discussed in Section 3. We then briefly illustrate Gong's protocol from [Go'93] in Section 4. After that we analyse a cross checksum scheme in Section 5. Based on this scheme we propose a key distribution protocol in Section 6. We conclude in Section 7.

2. Notation and Assumptions

We now review the main assumptions made in this paper. All protocols considered here are based on symmetric encipherment. We assume that two clients A and B wish to communicate securely with each other. For this purpose they need to verify the identity of one another and to establish a shared session key between them, but before the authentication processing starts they do not share any secret. Hence they need to make use of a third party, an authentication server S (or a set of servers $S_1, \ldots, S_n$).

In the protocols using a multiplicity of servers we also assume that:

◇ A and B do not trust any individual server,

◇ A and B believe that more than half the servers will correctly follow the protocol specifications, and

◇ A and B share secret keys K_{AS_i} and K_{BS_i} respectively with S_i.

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 also assume that hash functions used are collision resistant against 'off-line' search of pairs.

In the protocol descriptions, $A \rightarrow B : m$ indicates that A sends message m to B; $\{m\}_K$ denotes m encrypted with the key K; x, y denotes the concatenation of x and y; g and h are one-way hash functions; $\lfloor m \rfloor$ denotes the integer part of m; and $\lceil m \rceil$ denotes the smallest integer greater than or equal to m.

3. A Typical Protocol and an Associated Problem

We now discuss a typical protocol which uses symmetric encryption techniques and lets an authentication server S choose the session key.

This protocol is similar to a protocol in Clause 6.3 of ISO/IEC 11770-2 [11770-2]. In this protocol we suppose that A and S share a secret key K_{AS}, and B and S share another secret key K_{BS}.

$$\begin{aligned}
&M_1 : A \rightarrow B : A, B, N_A\\
&M_2 : B \rightarrow S : A, B, N_A, N_B\\
&M_3 : S \rightarrow B : \{A, N_B, K_{AB}\}_{K_{BS}}, \{B, N_A, K_{AB}\}_{K_{AS}}\\
&M_4 : B \rightarrow A : \{B, N_A, K_{AB}\}_{K_{AS}}, \{A, N_A, N'_B\}_{K_{AB}}\\
&M_5 : A \rightarrow B : \{B, N'_B, N_A\}_{K_{AB}}
\end{aligned}$$

where N_A, N_B and N'_B are three nonces, and K_{AB} is a session key for A and B.

A sends B a message M_1 containing a nonce N_A as a challenge; B then sends S a message M_2 including nonces N_A and N_B; S chooses a session key K_{AB} and distributes it to A and B in M_3; by checking the enclosed nonces, A and B verify that the reply of S is fresh and retrieve K_{AB}; finally A and B complete a handshake. After this protocol executes, A and B have agreed upon the session key K_{AB}, and A and B believe that K_{AB} has been retrieved by both clients and is appropriate for use between them.

A possible problem with this protocol is that A and B have to trust S in order to obtain the authentication information and the session key. S keeps total control over communications between A and B because S does all the 'verification of identities' and knows the session key. This protocol is vulnerable to active attacks by an 'untrustworthy third party'. That is, if S is corrupted, it can intercept communications between A and B, can impersonate A to B or B to A, and can leak A's and B's secrets.

4. Gong's Protocol with Multiple Servers

In order to solve the trust problem with the above protocol, Gong [Go'93] proposed a different protocol, again based on symmetric encryption, but with n servers $S_1, \ldots, S_n$ instead of one server. Each server is responsible for converting and distributing a part of the session key. In the following protocol, the steps M_{1i}, M_{2i}, M_{4i} and M_{5i} are repeated for $i = 1, \ldots, n$. Steps M_{1i} and M_{2i} must all be completed before step M_3 is performed; similarly steps M_{4i} and M_{5i} must all be completed before step M_6.

$$\begin{aligned}
M_{1i} &: A \rightarrow S_i : A, B \\
M_{2i} &: S_i \rightarrow A : N_{S_i} \\
M_3 &: A \rightarrow B : A, B, N_A, \\
&\qquad N_{S_1}, \{A, B, N_{S_1}, x_1, C(x)\}_{K_{AS_1}}, \ldots, \\
&\qquad N_{S_n}, \{A, B, N_{S_n}, x_n, C(x)\}_{K_{AS_n}} \\
M_{4i} &: B \rightarrow S_i : A, B, N_A, N_B, \\
&\qquad \{A, B, N_{S_i}, x_i, C(x)\}_{K_{AS_i}}, \\
&\qquad \{B, A, N_{S_i}, y_i, C(y)\}_{K_{BS_i}} \\
M_{5i} &: S_i \rightarrow B : \{B, N_A, y_i, C(y)\}_{K_{AS_i}}, \\
&\qquad \{A, N_B, x_i, C(x)\}_{K_{BS_i}} \\
M_6 &: B \rightarrow A : \{B, N_A, y_1, C(y)\}_{K_{AS_1}}, \ldots, \\
&\qquad \{B, N_A, y_n, C(y)\}_{K_{AS_n}}, \\
&\qquad \{N_A\}_{K_{AB}}, N_B \\
M_7 &: A \rightarrow B : \{N_B\}_{K_{AB}}
\end{aligned}$$

A chooses a candidate session key x and computes $x_i = f_{t,n}(x, i)$ for each server S_i. Here $f_{t,n}$ is a threshold function [Ko'85] that produces n shadows of x in such a way that it is easy to recover x from any t shadows, but less than t shadows reveal no information about x. To compute $f_{t,n}(x)$, A chooses a random polynomial $p(x)$ of degree $t-1$ with $p(0) = x$. A then computes $x_i = f_{t,n}(x, i) = p(i)$, $i = 1, \ldots, n$. Due to the property of interpolation, given any t of the x_i's, B can easily determine $p(x)$ and recover $x = p(0)$ [Sh'79]. Following the same procedure, B chooses a candidate session key y and A can recover it from any t shadows. With less than t shadows, no information about x and y can be determined.

In this protocol, a cross checksum scheme is used to verify the legitimacy of the shadows [Bl'79]. Cross checksums for x and y are defined as $C(x) = \{g(x_1), \ldots, g(x_n)\}$ and $C(y) = \{g(y_1), \ldots, g(y_n)\}$ respectively, where g is a one-way hash function. The session key is computed by $K_{AB} = h(x, y)$, where h is a predetermined one-way hash function. In order to prevent A or B imposing the session key, the choice of h is limited; for example, it cannot be an exclusive-or operation. In this protocol, the total number of messages is $4n + 3$.

It was said in Gong's paper [Go'93] that in fact there is a major difficulty in letting servers be involved in choosing the session key, because clients do not have a secure communication channel for verification purposes before authentication completes, and thus they cannot easily reach an agreement on what they have received from which servers. As mentioned in Section 1, in some environments it is useful to let servers, not clients, choose a session key. In the next section, we propose a cross checksum scheme which is the basis for the new protocol given in this paper. Using this cross checksum scheme, all servers can participate in choosing candidate session keys, and two clients are able to verify them and use all 'good' candidate keys to compute a session key.

5. Cross Checksums of Candidate Keys

Cross checksum schemes were used by Gong [Go'93] and Klein et al. [KOB'95] in key distribution protocols. This section discusses a particular cross checksum scheme which can be used for authenticated key establishment. We ignore for the moment the issue of verifying the 'freshness' of the keys, i.e. we ignore the entity authentication issues. In the next section we describe an authentication protocol which uses this cross checksum technique, and thus provides verified session key establishment between A and B.

Algorithm 1. *The cross checksum scheme works as follows.*

1. *S_i generates candidate session keys K_{Ai} and K_{Bi}, and sends them encrypted under K_{ASi} and K_{BSi} to A and B respectively. If K_{Ai} and K_{Bi} are good candidate keys, they satisfy $K_{Ai} = K_{Bi}$.*
2. *B computes the check values $g(K_{Bi})$ of the candidate keys K_{Bi}, where g is a globally known one-way hash function, then performs the following calculations and sends $G'_B(1), \ldots, G'_B(n)$ to A.*

$$g'(K_{Bi}) = \begin{cases} g(K_{Bi}) & \text{if } B \text{ believes that it has received the candidate key } K_{B_i} \\ EM1 & \text{otherwise,} \end{cases}$$

$$G_B = g'(K_{B1}), \ldots, g'(K_{Bn}),$$

$$G'_B(i) = \begin{cases} \{G_B\}_{K_{B_i}} & \text{if } B \text{ believes that it has received the candidate key } K_{B_i} \\ EM2 & \text{otherwise,} \end{cases}$$

   where EM1 and EM2 are error messages.
3. *On receipt of $G'_B(1), \ldots, G'_B(n)$ from B, A computes the check values $g(K_{Ai})$ of the candidate keys K_{Ai}, then performs the following calculations and sends $G'_A(1), \ldots, G'_A(n)$ to B.*

$$g'(K_{Ai}) = \begin{cases} g(K_{Ai}) & \text{if } A \text{ believes that both } A \text{ and } B \text{ have received the same key } K_{A_i} = K_{B_i} \\ EM1 & \text{otherwise,} \end{cases}$$

$$G_A = g'(K_{A1}), \ldots, g'(K_{An}),$$

$$G'_A(i) = \begin{cases} \{G_A\}_{K_{A_i}} & \text{if } A \text{ believes that both } A \text{ and } B \text{ have received the same key } K_{A_i} = K_{B_i} \\ EM2 & \text{otherwise.} \end{cases}$$

A possible set of message exchanges is then as follows, where the first step M_{1i} is repeated for $i = 1, \ldots, n$. Steps M_{1i} must all be completed before step M_2.

$$M_{1i} : S_i \rightarrow B: \{K_{Bi}\}_{K_{BS_i}}, \{K_{Ai}\}_{K_{AS_i}}$$
$$M_2 : B \rightarrow A : \{K_{A1}\}_{K_{AS_1}}, \ldots, \{K_{An}\}_{K_{AS_n}}, G'_B(1), \ldots, G'_B(n)$$
$$M_3 : A \rightarrow B : G'_A(1), \ldots, G'_A(n)$$

We now show how A and B can use the exchanged information to agree on a session key. In order for this process to work, A and B must trust at least $\lfloor n/2+1 \rfloor$ of the servers to behave correctly, i.e. at least $\lfloor n/2+1 \rfloor$ of the pairs (K_{Ai}, K_{Bi}) satisfy $K_{Ai} = K_{Bi}$.

On receipt of M_{1i} (or waiting a time-out period if an expected message does not arrive), B firstly checks whether a message with correct syntax has been received from every server, and then creates a sequence $G_B = g'(K_{B1}), \ldots, g'(K_{Bn})$. Because g is a one-way hash function it does not disclose the secret of the corresponding candidate key. B now wants to show G_B to A. However it is necessary to guarantee that any server can read G_B but cannot modify it without being detected. The solution we propose here is for B to send A another sequence $G'_B(1), \ldots, G'_B(n)$.

Upon receiving M_2, A generates a similar sequence $G'_A(1), \ldots, G'_A(n)$ by checking whether both A and B have received the same keys. A firstly decrypts each $G'_B(i)$ using K_{Ai}. Because more than half the servers follow the protocol specifications correctly, at least $\lfloor n/2+1 \rfloor$ of the values K_{Ai} will be equal to the corresponding values K_{Bi}; A may not get all values G_B encrypted by B, but A can decrypt at least $\lfloor n/2+1 \rfloor$ copies of G_B; some copies may not be the same as others, but at least $\lfloor n/2+1 \rfloor$ copies must be same. Checking these copies, A eliminates the minority of different copies and keeps the majority, so long as the majority contains $> n/2$ elements. A then computes the values of $g(K_{Ai})$, and compares them with the values of $g(K_{Bi})$ included in G_B. Thereafter, A creates G_A and $G'_A(1), \ldots, G'_A(n)$.

On receipt of M_3, B decrypts each $G'_A(i)$ using K_{B_i}. Whether M_3 has been corrupted by untrustworthy servers or not, as long as at least $\lfloor n/2+1 \rfloor$ copies of G_A are the same and the majority value of G_A contains $> n/2$ values of $g(K_{A_i})$, B then compares the values of $g(K_{A_i})$ with $g(K_{B_i})$ and retrieves all candidate keys which A has retrieved. Thus A and B can share all 'good' candidate keys which are used to construct a session key $K_{AB} = h$(all the good candidate keys), where h is a pre-determined one-way hash function.

Theorem 2. *Using the above cross checksum scheme, A and B can successfully establish a session key K_{AB}, given the following assumptions.*

1. *m servers out of a total n servers follow the protocol specifications correctly, where $m \geq \lfloor n/2+1 \rfloor$.*
2. *A and B both correctly follow the protocol specifications.*
3. *The $n - m$ 'bad' servers can only do the following:*
   - ◇ *fail to send messages to either A or B, or send messages with the wrong syntax;*
   - ◇ *send different candidate keys to A and B;*

- *eavesdrop on $G'_A(1), \ldots, G'_A(n)$ and $G'_B(1), \ldots, G'_B(n)$; and*
- *modify $G'_A(1), \ldots, G'_A(n)$ or $G'_B(1), \ldots, G'_B(n)$ in transit between A and B.*

Note that the system will clearly fail to establish a session key if malicious entities (either dishonest servers or other third parties) interfere with communications between A and B (or between B and the honest servers). We therefore assume that A and B will request messages to be sent again until the protocol succeeds.

PROOF. Suppose that m is the number of 'good' servers, where $m \geq \lfloor n/2+1 \rfloor$, and j is the number of pairs (K_{Ai}, K_{Bi}) which are received by A and B and satisfy $K_{Ai} = K_{Bi}$.

1. On the above assumptions, and the property of the encipherment algorithms assumed in Section 2, j must satisfy $j \geq m \geq \lfloor n/2+1 \rfloor > n/2$.
2. If $G'_B(1), \ldots, G'_B(n)$ have not been modified by bad servers, A obtains j copies of G_B encrypted by B. A combination of the $n-j$ corrupted servers can only construct at most $n-j$ consistent values of an 'incorrect' G_B, because of the use of encryption and $n-j \leq n - \lfloor n/2+1 \rfloor = \lceil n/2-1 \rceil < n/2$.
3. Hence A retrieves $j > n/2$ values K_{Ai} which are also received by B. Furthermore G_A includes j check values and $n-j$ error messages EM1, and $G'_A(1), \ldots, G'_A(n)$ includes j copies of G_A encrypted by A and $n-j$ error messages EM2.
4. For the same reasons as mentioned in items 2 and 3, B obtains G_A generated by A and retrieves j copies of K_{Bi} which are also retrieved by A.

We arrive at the conclusion that A and B can obtain $> n/2$ pairs (K_{Ai}, K_{Bi}) satisfying $K_{Ai} = K_{Bi}$, which will be used to compute a resultant session key K_{AB}. □

Theorem 3. *The above cross checksum scheme allows a group of collaborating servers to learn the session key (or force A and B to agree on different session keys) if $m \leq \lceil n/2-1 \rceil$. If $\lceil n/2-1 \rceil < m < \lfloor n/2+1 \rfloor$ (i.e. $m = n/2$) then colluding servers can always prevent A and B agreeing on a session key.*

PROOF. We first suppose that $m \leq \lceil n/2-1 \rceil$, and hence $n-m \geq \lfloor n/2+1 \rfloor > n/2$.

Suppose that the m 'good' servers are $S_1, \ldots, S_m$, and the $n-m$ 'bad' servers are $S_{m+1}, \ldots, S_n$ (and that the 'bad' servers are all colluding). The n servers send n pairs (K_{Ai}, K_{Bi}) to A and B (which may not agree). B creates $G'_B(1), \ldots, G'_B(n)$ where

$$G'_B(i) = \{g'(K_{B1}), \ldots, g'(K_{Bn})\}_{K_{Bi}} \quad (1 \leq i \leq n).$$

The $n-m$ 'bad' servers then modify $G'_B(1), \ldots, G'_B(n)$ to $\Gamma'_B(1), \ldots, \Gamma'_B(n)$, where

$$\Gamma'_B(i) = \begin{cases} G'_B(i) & \text{if } 1 \leq i \leq m \\ \{EM1, \ldots, EM1, g(K_{A(m+1)}), \ldots, g(K_{An})\}_{K_{Ai}} & \text{if } m+1 \leq i \leq n. \end{cases}$$

There are two simple ways in which $S_{m+1}, \ldots, S_n$ can invalidate the protocol with this strategy. Firstly, if they follow the protocol correctly and put $K_{Ai} = K_{Bi}$ $(m < i \leq n)$, then, on receipt of $\Gamma'_B(1), \ldots, \Gamma'_B(n)$, A will deduce that the first m

candidate keys are bad and the last $n-m$ candidate keys are good (this will follow since $n-m > n/2$). A will then eliminate the 'bad keys' and keep the 'good keys' in $G'_A(1), \ldots, G'_A(n)$, where

$$G'_A(i) = \begin{cases} EM2 & \text{if } 1 \leq i \leq m \\ \{EM1, \ldots, EM1, g(K_{A(m+1)}), \ldots, g(K_{An})\}_{K_{Ai}} & \text{if } m+1 \leq i \leq n. \end{cases}$$

After receiving $G'_A(1), \ldots, G'_A(n)$, B believes that A has retrieved the same last $n-m$ candidate keys. A and B use these candidate keys to compute the final session key, which means that the $n-m$ colluding servers can obtain the session key.

Secondly, the 'bad servers' can choose $K_{Ai} \neq K_{Bi}$ $(m < i \leq n)$. By changing K_{Bi} to K_{Ai} in the appropriate places in the message from B to A (as above), and then changing them back in the return message, the 'bad servers' can force A and B into the situation where they believe they have agreed on a key, but in fact they have different keys; to make matters even worse, both the key held by A and the key held by B will be known to the colluding 'bad servers'.

Put simply, in both attacks the majority of 'bad' servers can shut out the minority of 'good' servers and either make A and B agree on different keys and/or make them agree on a key known to all the 'bad' servers.

Finally, if $\lceil n/2 - 1 \rceil < m < \lfloor n/2 + 1 \rfloor$ (i.e. $m = n/2$), then, by failing to follow the protocol correctly, the 'bad servers' can prevent the protocol ever operating correctly, even if they do not interfere with communications between A and B (or between B and the 'good' servers), by preventing a majority of candidate keys ever being established. □

6. The New Protocol

We now present a key distribution protocol based on the cross checksum method of the previous section. The protocol can be arranged in two different ways, depending on the flow of messages exchanged.

The first is a 'parallel' version. A initiates the protocol by sending a request to B in M_1.

$$M_1 : A \rightarrow B : A, B, N_A$$

B then contacts every server S_i to request a candidate session key K_i. The following steps M_{2i} and M_{3i} are repeated for $i = 1, \ldots, n$.

$$M_{2i} : B \rightarrow S_i : A, B, N_A, N_B$$
$$M_{3i} : S_i \rightarrow B : \{A, N_B, K_i\}_{K_{BS_i}}, \{B, N_A, K_i\}_{K_{AS_i}}$$

After receiving answers from the n servers (or waiting a time-out period if an expected message does not arrive), B organises two sequences $G_B = g'(K_1), \ldots, g'(K_n)$ and $G'_B(1), \ldots, G'_B(n)$, as defined in Algorithm 1, and then sends the following message to A.

$$M_4 : B \rightarrow A : \{B, N_A, K_1\}_{K_{AS_1}}, \ldots, \{B, N_A, K_n\}_{K_{AS_n}}, G'_B(1), \ldots, G'_B(n)$$

On receipt of M_4, A checks it and organises two similar sequences $G_A = g'(K_1), \ldots, g'(K_n)$ and $G'_A(1), \ldots, G'_A(n)$, as specified in Algorithm 1. A then sends $G'_A(1), \ldots, G'_A(n)$ to B in M_5. B checks it in the same way. A and B thus obtain the same good candidate keys to construct a session key $K_{AB} = h$(all the good candidate keys). The last two steps are a handshake to inform each other that the correct session key has been retrieved.

$$\begin{aligned} M_5 &: A \rightarrow B : G'_A(1), \ldots, G'_A(n), \{B, N_B, N'_A\}_{K_{AB}} \\ M_6 &: B \rightarrow A : \{A, N'_A, N_B\}_{K_{AB}} \end{aligned}$$

The second is a 'cascade' version. The authentication request generated by A and B is sent to n servers via a cascade chain from S_1 to S_n. Every server randomly and securely chooses a candidate session key which is sent to A and B via the same cascade chain. The message exchanges are as follows:

$$\begin{aligned} M_1 &: A \rightarrow B : A, B, N_A \\ M_2 &: B \rightarrow S_1 : A, B, N_A, N_B \end{aligned}$$

The following step is repeated for $1 \leq i < n$:

$$\begin{aligned} M_{i+2} : S_i \rightarrow S_{i+1} : & A, B, N_A, N_B, \\ & \{A, N_B, K_1\}_{K_{BS_1}}, \{B, N_A, K_1\}_{K_{AS_1}}, \ldots, \\ & \{A, N_B, K_i\}_{K_{BS_i}}, \{B, N_A, K_i\}_{K_{AS_i}} \end{aligned}$$

The last four steps are as follows:

$$\begin{aligned} M_{n+2} : S_n \rightarrow B &: \{A, N_B, K_1\}_{K_{BS_1}}, \{B, N_A, K_1\}_{K_{AS_1}}, \ldots, \\ & \quad \{A, N_B, K_n\}_{K_{BS_n}}, \{B, N_A, K_n\}_{K_{AS_n}} \\ M_{n+3} : B \rightarrow A &: \{B, N_A, K_1\}_{K_{AS_1}}, \ldots, \{B, N_A, K_n\}_{K_{AS_n}}, \\ & \quad G'_B(1), \ldots, G'_B(n) \\ M_{n+4} : A \rightarrow B &: G'_A(1), \ldots, G'_A(n), \{B, N_B, N'_A\}_{K_{AB}} \\ M_{n+5} : B \rightarrow A &: \{A, N'_A, N_B\}_{K_{AB}} \end{aligned}$$

In either version of this protocol, if at least one good K_i is random and fresh it is guaranteed that the session key K_{AB} is unpredictable and fresh. Because only A and B know all the good candidate session keys, the session key K_{AB} is known only to A and B and not to any server (as long as $n \geq 2$). Since K_{AB} results from the verification between A and B, it is verifiable for both A and B. No one among the n servers and the two clients can impose K_{AB}. So the session key K_{AB} in this protocol satisfies the four properties mentioned in Section 1, on the assumption that more than half the servers follow the protocol specifications correctly.

Note that the second version works on the assumption that no server refuses to cooperate. A major advantage of this version is that the number of messages is only $n + 5$. Another advantage is that the clients do not need to signal back to every server. However the disadvantage of the version is that it is possible for a server to 'break' the procedure either maliciously or by mistake. For example, if a server simply refuses to cooperate, authentication and key distribution cannot be completed. One solution is to use this protocol with a control mechanism which can detect any server refusing to provide service.

We summarize the properties of our protocol.

1. The numbers of messages in the two versions of the protocol are $2n + 4$ and $n+5$ respectively, compared to $2n+2$ messages in Gong's first protocol [Go'89] and $4n + 3$ messages in Gong's second protocol [Go'93].
2. The number of rounds in the first version is lower than in Gong's second protocol, in particular, client A does not make contact with the servers. It would be potentially useful in mobile telecommunications systems (e.g. A is a mobile user and communicates with a 'base station' B, and B communicates with service providers S_i).
3. Because the candidate session keys are chosen by servers and no individual client can impose the resultant session key, the choice of h is less limited than in Gong's second protocol; for instance, an exclusive-or operation can be used here, which cannot be used in Gong's protocol. Also, we do not require a secret sharing scheme in the construction of the session key.
4. A possible disadvantage of this protocol is the potential length of messages with $G'_A(1), \ldots, G'_A(n)$ and $G'_B(1), \ldots, G'_B(n)$. G_B (or G_A) will contain nt bits (given g outputs a t-bit value) and hence $G'_B(i)$ (or $G'_A(i)$) will contain at least this number of bits. Hence $G'_A(1), \ldots, G'_A(n)$ and $G'_B(1), \ldots, G'_B(n)$ will contain $> n^2t$ bits. However, for practical applications, a typical choice for t might be 200 and n might be 20. This gives a message length of approximately 10 kbytes, which is not a particularly large value. Moreover, the total size of messages here is less than in Gong's second protocol. The reason is that the size of G_A (G_B) is comparable with the size of $C(x)$ ($C(y)$), so the size of $G'_A(1), \ldots, G'_A(n)$ ($G'_B(1), \ldots, G'_B(n)$) is comparable with $\{C(x)\}_{K_{AS1}}, \ldots, \{C(x)\}_{K_{ASn}}$ ($\{C(y)\}_{K_{BS1}}, \ldots, \{C(y)\}_{K_{BSn}}$). In this protocol, such messages have to be transmitted once; however, in Gong's protocol, such messages are transmitted three times.

7. Conclusions

Key distribution protocols without the assumption of trusting an individual authentication server are needed in some environments where clients have no reason to trust individual servers. A cross checksum scheme for the verification of candidate keys is analysed in this paper. These candidate keys are generated by multiple servers in an environment where no individual server is trusted. A protocol with an arbitrary number of authentication servers using the cross checksum scheme is proposed here. On the assumption that more than half the servers follow the protocol specifications correctly, four desirable properties about the session key are guaranteed. The session key is (1) unpredictable and fresh, (2) known only to clients and not to any server, (3) verifiable for every client, and (4) not imposed by any individual client or server. A minority of corrupted and colluding servers cannot compromise security in either version of the protocol, and cannot break the procedure in the parallel version of the protocol. The number of exchanged messages and the size of total messages in the first version of the protocol are lower than the protocol proposed by Gong [Go'93] with the same highly secure and available properties. The number of messages in the second is lower than in the first, with the same property of high security.

Possible future topics for research in this area include the following.

- Can efficient protocols be designed for the case where $n/2$ or fewer of the authentication servers are trustworthy?
- As is common in the design and analysis of distributed protocols, it would be useful to distinguish between failed authentication servers, which fail to take part in protocols, and corrupted authentication servers, which participate in a dishonest way. Protocols could then be designed to deal with various proportions of failed and dishonest servers.
- The derivation of lower bounds on the number of messages in various types of protocol would give a measure on how efficient specific protocols are.
- It would be of interest to see if more efficient protocols could be designed based on the use of timestamps and synchronised clocks (as is normally the case).

Acknowledgements

The authors would like to acknowledge the valuable comments and encouragement of their colleagues in the UK DTI/EPSRC LINK PCP 3GS3 project. The anonymous referees are also to be thanked for a number of important corrections and clarifications.

References

[Bl'79] G.R. Blakley, "Safeguarding cryptographic keys", in *Proceedings of AFIPS 1979 NCC, Vol. 48, Arlington, Va.*, 1979.

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

[DiHe'76] W. Diffie and M.E. Hellman, "New directions in cryptography", *IEEE Transactions on Information Theory* **22**, (1976), 644–654.

[Go'89] L. Gong, "Securely replicating authentication services", pp. 85–91 in *Proceedings: IEEE 9th International Conference on Distributed Computing Systems*, Newport Beach, California, 1989.

[Go'93] L. Gong, "Increasing availability and security of an authentication service", *IEEE Journal on Selected Areas in Communications* **11**, (1993), 657–662.

[11770-2] ISO/IEC, *11770-2*, Information technology — Security techniques — Key management — Part 2: Mechanisms using symmetric techniques, (1996).

[KOB'95] B. Klein, M. Otten, and T. Beth, "Conference key distribution protocols in distributed systems", pp. 225–241 in *Codes and Cyphers, Proceedings of the Fourth IMA Conference on Cryptography and Coding* (P. G. Farrell, ed.), Formara Limited. Southend-on-sea. Essex, 1995.

[Ko'85] S.C. Kothari, "Generalized linear threshold scheme", pp. 231–241 in *Advances in Cryptology: Proc. Crypto '84* (G. R. Blakley and D. Chaum, ed.), Lecture Notes in Computer Science 196, Springer-Verlag, 1985.

[Me'90] R.C. Merkle, "A fast software one way hash function", *Journal of Cryptology* **3(1)**, (1990), 43–58.

[Pe'91] T.P. Pedersen, "A threshold cryptosystem without a trusted party", pp. 522–526 in *Advances in Cryptology: Proc. Eurocrypt '91* (D. W. Davies, ed.), Lecture Notes in Computer Science 547, Berlin: Springer-Verlag, 1991.

[Sh'79] A. Shamir, "How to share a secret", *Communications of the ACM* **22**, (1979), 612–613.

[YKB'93] 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).