Key Distribution without Individual Trusted Authentication Servers*

L. Chen, D. Gollmann and C. Mitchell
Information Security Group
Royal Holloway, University of London
Egham, Surrey TW20 0EX, 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, which have the property that a minority of them may be untrustworthy. This paper proposes two key distribution protocols with multiple authentication servers using a cross checksum scheme. Both protocols are based on the use of symmetric encryption for verifying the origin and integrity of messages. In these protocols it is not necessary for clients to trust an individual authentication server. A minority of malicious and colluding servers cannot compromise security and can be detected. The first 'parallel' protocol can prevent a minority of servers disrupting the service. The second 'cascade' protocol has to work with other security mechanisms in order to prevent a server breaking the procedure by refusing to cooperate. As compared with other proposed protocols with similar properties these two protocols require less exchanged messages.

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. Typically this third party is an authentication server who 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. Unfortunately not every authentication server is always trustworthy. If the authentication server is malicious, or is compromised, the security of communications between these clients cannot be guaranteed.

In order to make a protocol work in an environment where clients do not trust an individual server, it is important to find authentication schemes which reduce the requirement for trusting servers. Some recent research [1, 2, 3, 6, 8] 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 in which the use of a more complex authentication service results in a more secure and available authentication service have been considered.

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 [8] 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. Gong [2] proposed a protocol with multiple authentication servers such that a minority of malicious 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.

However in some environments it is necessary to let servers, not clients, choose a session key, for example, in environments where clients cannot randomly and securely generate a session key or candidate keys. One potential problem with letting servers be involved in choosing candidate session keys is 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 it is received by both clients. In order to solve this problem, we discuss a cross checksum scheme, which works under 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 participate in verifying these keys by exchanging the relevant check values using a one-way globally known hash function in order to check whether every candidate key is received by both clients correctly. These clients

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

then eliminate bad candidate keys and use good keys to compute a final session key. It is not necessary for clients to trust any individual server because no single server can know the session key.

Based on this scheme we propose two key distribution protocols with an arbitrary number of authentication servers. In these protocols a minority of malicious and colluding authentication servers cannot compromise security and can be detected. The first 'parallel' protocol can prevent a minority of servers disrupting the service. The second 'cascade' protocol has to work with other security mechanisms in order to prevent a server breaking the procedure by refusing to cooperate. If more than half the servers follow the protocol specifications correctly, the following properties will hold for the session key:

- ◇ it will be fresh (i.e. not a replay of an old key) and random (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 has not been chosen by any individual client or server.

The two protocols with n servers we discuss in this paper use $2n+4$ and $n+5$ messages, whereas Gong's protocol [2] uses $4n+3$ messages.

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 [2] in Section 4. After that we analyse a cross checksum scheme in Section 5. Based on this scheme we propose two key distribution protocols, one of which is a parallel protocol specified in Section 6 and the other is the cascade protocol in Section 7. We conclude in Section 8.

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 currently share any secret. So they make use of a third party, an authentication server S (or a set of servers S_1, ..., 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 more than half the servers will correctly follow the protocol specifications,
- ◇ 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 possessor of the secret in the form that it was received.

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; and $\lfloor m \rfloor$ denotes the integer part of 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 DIS 11770-2 [5]. 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}.

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

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 session key K_{AB}, and A and B believe that K_{AB} has been retrieved by each other 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 'untrustworthy third party' attacks. That is, if S is malicious, he 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 this problem with the above protocol, Gong [2] proposed a different protocol, again based on symmetric encryption, but with n servers S_1,...,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 $i = 1, ..., n$. 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 completed before step M_6.

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

A (or B) chooses a candidate session key x (or y) and computes $x_i = f_{t,n}(x, i)$ (or $y_i = f_{t,n}(y, i)$) for each server S_i. Here $f_{t,n}$ is a threshold function [4] that produces n shadows of x (or y) in such a way that it is easy to recover x (or y) from any t shadows, but less than t shadows reveal no information about x (or y). To compute $f_{t,n}(x)$ (or $f_{t,n}(y)$), A (or B) chooses a random polynomial $p(x)$ of degree $t-1$ (or $p(y)$) with $p(0) = x$ (or $p(0) = y$). A (or B) then computes $x_i = f_{t,n}(x, i) = p(i)$ (or $y_i = f_{t,n}(y, i) = p(i)$), $i = 1, ..., n$. Due to the property of interpolation, given any t of the x_i's (or y_i's), B (or A) can easily determine $p(x)$ (or $p(y)$) and recover $x = p(0)$ (or $y = p(0)$) [7]. With less than t shadows, no information about x (or y) can be determined.

In this protocol, a cross checksum scheme is used to verify the legitimacy of the shadows. Cross checksums for x and y are defined as $C(x) = \{g(x_1), ..., g(x_n)\}$ and $C(y) = \{g(y_1), ..., g(y_n)\}$, 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 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 two new protocols 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 [2] and Klein et al [3] 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 two sections we describe authentication protocols which use this cross checksum technique, and thus provide 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 $G'_B(1), \cdots, G'_B(n)$ to A.*

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

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

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

where EM1 and EM2 are error messages.

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

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

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

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

A possible set of message exchanges is then as follows, where the first step M_{1i} is repeated for $i = 1, ..., n$.

Steps M_{1i} must all be completed before step M_2.

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

We now show how A and B can use the exchanged information to agree 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 the message with a correct syntax from every server has been received, and then creates a sequence $G_B = g'(K_{B1}), ..., g'(K_{Bn})$. 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. One solution is for B to send A another sequence $G'_B(1), ..., G'_B(n)$.

After receiving M_2, A generates a similar sequence $G'_A(1), ..., G'_A(n)$ by checking whether both A and B have received the same key. A firstly decrypts each $G'_B(i)$ using K_{Ai}. Because more than half the servers follow the protocol specifications correctly, some K_{Ai} may not be the same as K_{Bi}, but at least $\lfloor n/2+1 \rfloor$ of the values K_{Ai} are equal to the corresponding values K_{Bi}; A may not get all G_B encrypted by B, but A can decrypt at least $\lfloor n/2+1 \rfloor$ G_B copies; 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. After that A creates G_A and $G'_A(1), ..., G'_A(n)$.

On receipt of M_3, B checks it in the same way. Finally A and B retrieve 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 retrieve all good candidate session keys, and the number of these keys is at least $\lfloor n/2 + 1 \rfloor$, given the following four assumptions:*

1. *There are m servers out of a total n servers following the protocol specifications correctly, $m \geq \lfloor n/2 + 1 \rfloor$.*
2. *A and B both correctly follow the protocol specifications.*
3. *The $n - m$ bad servers can possibly do the following:*
   - ◇ *not send messages to A and B or send messages with the wrong syntax;*
   - ◇ *send different candidate keys to A and B;*
   - ◇ *eavesdrop on $G'_A(1), ..., G'_A(n)$ and $G'_B(1), ..., G'_B(n)$; and*
   - ◇ *modify $G'_A(1), ..., G'_A(n)$ or $G'_B(1), ..., G'_B(n)$ in transit between A and B.*

Note that the system will clearly fail 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 K_{Ai} K_{Bi} pairs which are received by A and B, and satisfy $K_{Ai} = K_{Bi}$.

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

We arrive at the conclusion that A and B can obtain at least $\lfloor n/2 + 1 \rfloor$ pairs K_{Ai} K_{Bi} which satisfy $K_{Ai} = K_{Bi}$. □

Theorem 3 *The above cross checksum scheme does not work if $m < \lfloor n/2 + 1 \rfloor$.*

Proof: We describe a possible attack. Suppose $m = \lfloor (n-1)/2 \rfloor$, e.g. there are $\lfloor (n-1)2 \rfloor$ good servers $S_1, ..., S_{\lfloor (n-1)/2 \rfloor}$ and $\lfloor n/2+1 \rfloor$ bad servers $S_{\lfloor (n+1)/2 \rfloor}$, ..., S_n who are colluding. These n servers send n pairs of K_{Ai} K_{Bi} to A and B. B creates $G'_B(1), ..., G'_B(n)$ where

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

The $\lfloor n/2 + 1 \rfloor$ bad servers then modify $G'_B(1)$,..., $G'_B(n)$ to $\tilde{G}'_B(1), \cdots, \tilde{G}'_B(n)$, where

$$\tilde{G}'_B(i) = \begin{cases} G'_B(i) & i = 1, ..., \lfloor (n-1)/2 \rfloor \\ \{EM1, \cdots, EM1, g(K_{B\lfloor (n+1)/2 \rfloor}), \cdots, g(K_{Bn})\}_{K_{Bi}} & i = \lfloor (n+1)/2 \rfloor, ..., n. \end{cases}$$

From the above message, A derives the wrong result that the first $\lfloor (n-1)/2 \rfloor$ candidate keys are bad and the last $\lfloor n/2+1 \rfloor$ candidate keys are good, and then eliminates 'bad keys' and keeps 'good keys' in $G'_A(1), \cdots, G'_A(n)$, where

$$G'_A(i) = \begin{cases} EM2 & i = 1, ..., \lfloor (n-1)/2 \rfloor \\ \{EM1, \cdots, EM1, g(K_{A\lfloor (n+1)/2 \rfloor}), \cdots, \\ \quad g(K_{An})\}_{K_{Ai}} & \\ & i = \lfloor (n+1)/2 \rfloor, ..., n. \end{cases}$$

After receiving $G'_A(1), \cdots, G'_A(n)$, B believes that A has retrieved the same last $\lfloor n/2+1 \rfloor$ candidate keys. A and B use these candidate keys to compute the final session key, which means that the $\lfloor n/2+1 \rfloor$ colluding servers can obtain the session key.

If $m < \lfloor (n-1)/2 \rfloor$, the above attack is still valid.

If $m = n/2$ when n is even, A (or B) may find it difficult to make a decision about which candidate keys should be kept or eliminated.

Thus the cross checksum scheme does not work if $m < \lfloor n/2+1 \rfloor$. □

6 A parallel protocol

In this section we present a key distribution protocol based on the cross checksum method of the previous section. A initiates the protocol by sending a request to B in M_1.

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

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

$$\begin{aligned} M_{2i}: \quad & B \rightarrow S_i: \quad A, B, N_A, N_B \\ M_{3i}: \quad & S_i \rightarrow B: \quad \{A, N_B, K_i\}_{K_{BS_i}}, \\ & \qquad\qquad \{B, N_A, K_i\}_{K_{AS_i}} \end{aligned}$$

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

$$\begin{aligned} M_4: \quad B \rightarrow A: \quad & \{B, N_A, K_1\}_{K_{AS_1}}, \cdots, \\ & \{B, N_A, K_n\}_{K_{AS_n}}, \\ & G'_B(1), ..., G'_B(n) \end{aligned}$$

On receipt of M_4, A checks it and organises two similar sequences $G_A = g'(K_1), ..., g'(K_n)$ and $G'_A(1), ..., G'_A(n)$, as specified in Algorithm 1. A then sends $G'_A(1), ..., G'_A(n)$ 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.

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

In this protocol, if at least one good K_i is random and fresh it is guaranteed that the session key K_{AB} is random 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.

As compared with Gong's protocol with similar properties, this protocol has the following advantages.

1. The number of messages in this protocol is $2n+4$ which is lower than $4n+3$;
2. Because the candidate session keys are chosen by servers, the choice of $h()$ is less limited than in Gong's protocol; for instance, an exclusive-or operation can be used here, which cannot be used in Gong's protocol; and
3. The computational complexity of this protocol is lower because no polynomial interpolation is used.
4. A possible disadvantage of this protocol is the potential length of messages M_4 and M_5. 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 M_4 and M_5 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 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 $G'_A(1), ..., G'_A(n)$ ($G'_B(1), ..., G'_B(n)$) is comparable with $\{C(x)\}_{K_{AS1}}, ..., \{C(x)\}_{K_{ASn}}$ ($\{C(y)\}_{K_{BS1}}, ..., \{C(y)\}_{K_{BSn}}$). In this protocol, such messages have to be transmitted once, however, in Gong's protocol, such messages are transmitted three times.

7 A cascade protocol

In this section we consider a second 'cascade' key distribution protocol again based on the cross checksum scheme of Section 5. Note that this protocol works on the assumption that no server refuses to provide a service. The protocol is as follows:

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

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

$$M_{i+2}: \quad S_i \rightarrow S_{i+1}: \quad A, B, N_A, N_B, \{A, N_B, K_1\}_{K_{BS_1}}, \{B, N_A, K_1\}_{K_{AS_1}}, \cdots, \{A, N_B, K_i\}_{K_{BS_i}}, \{B, N_A, K_i\}_{K_{AS_i}}$$

The last four steps are as follows:

$$M_{n+2}: \quad S_n \rightarrow B: \quad \{A, N_B, K_1\}_{K_{BS_1}}, \{B, N_A, K_1\}_{K_{AS_1}}, \cdots, \{A, N_B, K_n\}_{K_{BS_n}}, \{B, N_A, K_n\}_{K_{AS_n}}$$

$$M_{n+3}: \quad B \rightarrow A: \quad \{B, N_A, K_1\}_{K_{AS_1}}, \cdots, \{B, N_A, K_n\}_{K_{AS_n}}, G'_B(1), ..., G'_B(n)$$

$$M_{n+4}: \quad A \rightarrow B: \quad G'_A(1), ..., G'_A(n), \{B, N_B, N'_A\}_{K_{AB}}$$

$$M_{n+5}: \quad B \rightarrow A: \quad \{A, N'_A, N_B\}_{K_{AB}}$$

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. Using an analysis similar to that in the previous section we can show that A and B can detect and eliminate every bad candidate session key which is not received by A and B correctly. A and B compute K_{AB} by using all the good candidate keys. Then A and B complete a two way handshake.

As in the previous analysis, in this protocol the session key K_{AB} also satisfies the four properties mentioned in Section 1, on the assumption that more than half the servers follow the protocol specifications correctly. That means this protocol is as secure as the previous protocol. A major advantage of the protocol 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 protocol 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, thus ensuring that no one can break the procedure.

8 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 which no individual server is trusted but more than half of them are believed to behave correctly. Two protocols with an arbitrary number of authentication servers using the cross checksum scheme are 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) random and fresh, (2) to be known only to clients and not to any server, (3) verifiable for every client, and (4) not to be imposed by any individual client or server. A minority of malicious and colluding servers cannot compromise security in either protocol, and cannot break the procedure in the first protocol. The computational complexity, the number of exchanged messages and the size of total messages in the first protocol are lower than the protocol proposed by Gong [2] with the same highly secure and available properties. The computational complexity and message number of the second protocol are lower than 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 $\leq n/2$ 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 malicious 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

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

[2] L. Gong. Increasing availability and security of an authentication service. *IEEE Journal on Selected Areas in Communications*, 11(5):657–662, June 1993.

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

[4] S.C. Kothari. Generalized linear threahold scheme. In *Lecture Notes in Computer Science 196, Advances in Cryptology: Proc. Crypto '84*, pages 231–241. Springer-Verlag, 1985.

[5] ISO/IEC JTC 1/SC 27 N972. ISO/IEC DIS 11770-2, Information technology — Security techniques — Key management — Part 2: Mechanisms using symmetric techniques. November 1994.

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

[7] A. Shamir. How to share a secret. *Communications of the ACM*, **22**(11):612–613, November 1979.

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