

A Class of Flexible and Efficient Key Management Protocols

Colin Boyd
Information Security Research Centre
School of Data Communications
Queensland University of Technology
Brisbane Q4001
Australia

Email: boyd@fit.qut.edu.au

Abstract

Cryptographic protocols for key establishment normally include some means to allow participants to ensure that a key is new and not replayed from an old protocol run. When the key is generated by a mutually trusted server this is usually achieved by sending with the key a quantity known to be new. A different general method for achieving freshness in this context is proposed. A number of specific example protocols are given which have some practical advantages over previous published protocols.

1. Introduction

Protocols for authentication and key establishment are used to set up secure communications sessions. Protocols using symmetric cryptographic techniques usually involve a mutually trusted party who shares keys with the parties which will be involved in the session. The trusted party is often responsible for generation of the temporary session key. When a user receives a key from a trusted party what he needs to know, apart from the key's value, is:

- which other users know that key.
- that the key is fresh for this session.

The first of these properties is usually established by receiving the names of the other recipients of the key in an authenticated message from a trusted server. The second is usually verified by checking the freshness of some item linked with the key. Typical methods are one of the following two.

Challenge-Response The key recipient generates a random number and sends it to the server. This *challenge*

is returned with the key. Since the recipient knows that the challenge is fresh he deduces that the key is also fresh.

Timestamps When the key is sent by the server the current time is included. The recipient checks that the timestamp is in a certain window of its current clock time.

Timestamps are problematic in applications involving widely distributed systems because of the difficulties of clock synchronisation and so the challenge-response method is often preferred.

Gong [7] has published a study of protocol efficiency concerns. The number of messages contained in a protocol is a measure of the number of separate events that must be initiated during a protocol execution. In many cases different messages may be sent in parallel. Gong defines a *round* to consist of all messages that can be sent and received in parallel within one time unit. The number of rounds required by a protocol is a measure of the length of time that a protocol execution must take. Gong analysed and produced lower bounds for the number of messages and rounds that must be used in various types of secure protocols. In general, protocols that minimise the number of messages do *not* also minimise the number of rounds.

In this paper a method of establishing key freshness is proposed which has largely been ignored in protocols of this type. General application of the method results in new protocols which have a number of potential advantages over previously published protocols; in particular it is possible to lower Gong's lower bound for protocols in similar scenarios (although it should be emphasised that a different model for security is used). Another useful (and unusual) property is that it is possible to 'cache' replies from the server for use in later protocol executions.

2. Novel Method for Freshness

2.1. Freshness by Random Input

The usual methods for establishing freshness of a key have been outlined above. These rely on checking the freshness of an element *received* with the key. An alternative way to ensure freshness is for the key user to generate a fresh input to the key generation process. Now this method is not new, but has been used where the key users themselves are responsible for key generation (usually called *key agreement* [12]). The new idea is that this mechanism can still be used as a general method for obtaining freshness even when a server remains responsible for maintaining confidentiality of the key - that is, who may get the key. (In general, it is possible to divide most protocols into one of four classes depending on whether the users establish freshness by input or receipt, and on whether they control who gets the key or not. This idea is explored further in a related paper [5].)

To use the idea the session key generated must be a function f of several inputs. This function must have the right properties to ensure that the security requirements are satisfied.

1. In order to maintain confidentiality of the session key at least one input of f must be confidential to the protocol participants. f should then have the property that, without knowledge of this input, the value of f cannot be calculated whatever the values of the other inputs.
2. To achieve freshness for each participant f must have the property that if each participant's input is fresh then it is infeasible to choose the other inputs so that the output of f is an old value.

It is possible to proceed to design protocols by simply specifying that the function used should have the desired properties. Luckily it turns out that suitable practical functions exist on which to base concrete protocols. These base functions are variously called *keyed hash-functions* or *message authentication codes (MACs)*. There has been considerable interest lately in the design of such functions [3, 1] but as yet there appears to be no consensus on the exact properties that they should have¹ and so it is useful to summarise here the properties required for the protocols of interest.

A *keyed hash function* (KHF) is a mapping $f : K \times M \rightarrow C$ where K is a space of *hashing keys* of a fixed size, M is the space of all binary strings of any finite size, and C is the space of all strings of a fixed size (typically 128 or 160 bits). In addition the following security properties hold.

¹This may well be because the exact properties required vary in different application scenarios.

KHF1(Practicality) Given any k and m it is easy to calculate $f(k, m)$.

KHF2 (Confidentiality) Without knowledge of k it is infeasible to calculate $f(k, m)$ for any m , even given the value of many $f(k, m_i)$ (with $m_i \neq m$).

KHF3 (Freshness) Given k it is infeasible to find any pair of messages m_1 and m_2 such that $f(k, m_1) = f(k, m_2)$.

Note that KHF1 and KHF2 together imply that it is infeasible to find the hashing key k given many $f(k, m_i)$ values. The names given to the above properties correspond to the security properties for which they will be used in protocols and are not the usual names given to them. In particular, KHF3 is usually called *collision resistance*, but the need for it in the protocols is that it should be infeasible ever to force an old key to be re-used - in other words, by ensuring that the input is different from before, any participant can ensure the key is fresh.

The definition of KHF above is very similar to that of Bakhtiari, Safavi-Naini and Pieprzyk [1], and is strictly weaker than that of Berson, Gong and Lomas [3]. Of practical importance it should be noted that both the papers cited give constructions to obtain concrete KHFs, which can equally be used to implement the protocols discussed in this paper.

2.2. General Methodology

The general design for protocols can now be simply explained. The session key will be derived as the output of a KHF. The hashing key for the KHF could be shared beforehand but in general this will not be the case and confidentiality of the session key will be derived from secure transport of the hashing key to all participants. The hashing key could be chosen by any participant depending on the scenario; below the idea is illustrated with the conventional scenario of a trusted server who chooses the secret.

The novel aspect is the method of ensuring freshness. The session key SK key will be defined as

$$SK = f(k, N_1|N_2|\dots|N_r)$$

where f is a KHF and $N_1|N_2|\dots|N_r$ is the concatenation of the inputs by each of the r participants. The definition of the KHF guarantees that SK has the required properties as follows.

- KHF1 guarantees that all participants can efficiently derive SK .
- As long as k is known to be shared by only the participants (possibly in addition to one or more trusted servers) then KHF2 guarantees that SK is confidential to the participants.

- As long as each participant knows that at least one of N_1, N_2, \dots, N_r is new then KHF3 guarantees the freshness of SK .

A distinctive aspect of this general design is that session key confidentiality and session key freshness are derived independently. This leads to considerable flexibility in using such protocols and a number of useful properties not shared by other published protocols. This point is discussed further below.

3. A New Protocol

The following protocol presents a typical example of the methodology. The general format is typical for key management protocols using symmetric cryptography and a trusted server S . The two protocol users A and B initially share keys, K_{AS} and K_{BS} respectively, with S . The aim of the protocol is to allow A and B to establish a key K_{AB} which they are confident is a new key only known to themselves and S .

1. $A \rightarrow S : A, B, N_A$
2. $S \rightarrow B : \{A, B, K_S\}_{K_{AS}}, \{A, B, K_S\}_{K_{BS}}, N_A$
3. $B \rightarrow A : \{A, B, K_S\}_{K_{AS}}, [N_A]_{K_{AB}}, N_B$
4. $A \rightarrow B : [N_B]_{K_{AB}}$

Both A and B need to calculate the value of K_{AB} by

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

where f is a previously agreed KHF. In this description distinction is made between the cases where the encryption function used is needed to provide confidentiality and when it is needed to provide information integrity. The notation $\{M\}_K$ denotes that the message M is encrypted using key K and the encryption function used provides both confidentiality (to prevent the value K_S from being discovered by an eavesdropper) and integrity (to prevent the identities of the users being altered). The notation $[M]_K$, by contrast, denotes that M is transformed only to provide integrity, but since M may sometimes contain information that should not be revealed it is also assumed that it is not feasible to obtain M from $[M]_K$. In other words, $[M]_K$ may be regarded as an integrity check value on M .

3.1. Security

The general form for the protocol has been designed to ensure that the chosen security requirements are achieved. Nevertheless it is worthwhile to consider carefully whether this is really the case for the concrete protocol reached.

This is especially true because the protocol appears, superficially, to be very similar to some protocols with known flaws.

The first thing to note is that security may be considered with respect to each user separately; indeed the protocol is essentially symmetric with respect to A and B . Consider A 's viewpoint. She receives the message $\{A, B, K_S\}_{K_{AS}}$ from S (its physical routing via B is of no relevance). As long as the encryption provides integrity of data (an assumption it is important to make explicit) A can be sure that S intended K_S to be shared between A and B . The encryption must also provide confidentiality so that other users cannot gain access to this value.

Since the message from the server carries no information to show that it is new, an obvious method of possible attack is to exploit the possibility of replaying an old session key K_{AB} which has been compromised. It is a normal assumption in protocol analysis that such an attack is possible. Let us consider what an attacker may gain from this. Since each user is able to gain independent assurance that each key used is fresh there is no chance for the attacker to force the use of an old key. The only benefit he may be able to gain is to use knowledge of an old key to aid a cryptanalytic attack. If the attacker could gain knowledge of K_S this would be sufficient for him to be able to obtain the new keys shared by A and B . Thus K_S must remain as secure as the shared keys K_{AS} and K_{BS} ². Furthermore, the cryptographic strength of the KHF f should be equal to that of the encryption function used by the server.

The security properties of the protocol can be summarised in the following result, subject to the assumption (normal for cryptographic protocols) that the cryptographic algorithms used are not subject to cryptanalysis.

Claim 1 *At the end of a successful protocol run Alice and Bob can both be sure that the session key is new and shared only with the other and the trusted server.*

3.2. Protocol Properties

The protocol given above could be claimed as of interest in its own right as the product of a new design method. However it turns out to have some remarkable properties which give it distinct advantages over previously known examples in a variety of situations.

3.2.1 Number of messages and rounds

The protocol above has four messages. Since each message after the first relies on information received in the previ-

²Note that at no time is K_S used to encrypt any value. Furthermore, if K_S is stored for future use it may be safely left in the encrypted form as received from S so that there is no additional requirement for secure storage over more conventional schemes.

ous message it follows that the protocol also requires four rounds. However, it is possible to re-route some of the information to form a related protocol with more messages but fewer rounds as follows.

1. $A \rightarrow S : A, B$
2. $A \rightarrow B : A, N_A$
3. $S \rightarrow B : \{A, B, K_S\}_{K_{BS}}$
4. $S \rightarrow A : \{A, B, K_S\}_{K_{AS}}$
5. $B \rightarrow A : B, N_B$
6. $B \rightarrow A : [N_A]_{K_{AB}}$
7. $A \rightarrow B : [N_B]_{K_{AB}}$

As before A and B calculate the value of K_{AB} by $K_{AB} = f(K_S, N_A | N_B)$. Although this variant has an extra three messages it is now possible to send messages 1 and 2 in parallel, 3, 4, and 5 in parallel, and 6 and 7 in parallel thus reducing the number of rounds to three.

Gong [7] has derived tight lower bounds for the number of messages and rounds for protocols in a variety of classes. One criterion used for classification is who generates the session key. Gong considers situations where the server alone chooses it, and where one or both users generate it. He does not consider that all three might take part.

The other criterion used by Gong is whether freshness is achieved by timestamps or by nonces (random challenges). Of course the new protocol does not properly fall into either category but is much nearer to the second in that both rely on the generation of random numbers and do not rely on clock synchronisation.

Gong's lower bounds for protocols in which nonces are used, and where a handshake is included, are 5 messages and 4 rounds if the server chooses the key, or 6 messages and 5 rounds if the two users choose the key. Since we have demonstrated protocols lowering these bounds we arrive at the following result.

Claim 2 *Protocols using the new method for freshness can be designed to be more efficient in the number of rounds and the number of messages than any protocol using the conventional challenge-response method.*

It is not difficult to see that the reason this efficiency is achieved is that the server does not have to wait for nonces from both users before generating and distributing the 'key'. This feature is common with protocols that rely on timestamps for freshness and so it is not surprising that the lower bounds for this new class of protocols are the same as Gong has found for protocols using timestamps. In a sense it is reasonable to say that the new method gives the convenience

of using timestamps without introducing the problems of clock synchronisation. Further advantages are now considered.

3.2.2 Re-use of keying material

To the reader familiar with similar protocols, one of the most striking features of the protocol above will be that the messages received from S by A and B are independent of the input from A and B . Of course the whole point is that freshness does not come from the server's input, but an unexpected consequence is that users may save much computation.

Certain authors [10, 9] have noted that it may sometimes be desirable to renew authentication during a session without contacting the server again; those same authors have given protocols to achieve this. With the new protocol above re-authentication can be achieved in an obvious and simple manner. A and B simply choose new random values N'_A and N'_B and use these to define the new key $K'_{AB} = f(K_S, N'_A | N'_B)$. This may then be followed by a handshake to establish that the other has the key.

This feature may also be exploited to delay authentication and key exchange until a later time without using the server. Consider the following variant protocol.

1. $A \rightarrow S : A, B$
2. $S \rightarrow A : \{A, B, K_S\}_{K_{BS}}, \{A, B, K_S\}_{K_{AS}}$
3. $A \rightarrow B : A, B, \{A, B, K_S\}_{K_{BS}}, N_A$
4. $B \rightarrow A : [N_A]_{K_{AB}}, N_B$
5. $A \rightarrow B : [N_B]_{K_{AB}}$

Here messages 1 and 2 may be exchanged at any time before the other messages and without knowledge of B . This means that A may cache messages from S to use at any later time. A may also use these cached messages multiple times. Note that there is no security problem with this since A and B insert new inputs each time to give key freshness. We can regard the values $\{A, B, K_S\}_{K_{BS}}$ and $\{A, B, K_S\}_{K_{AS}}$ as renewable tokens for A and B .

As long as f is a good KHF, there is no reason to suppose that K_S will become any more vulnerable than the long-term keys K_{AS} and K_{BS} even with repeated use. Of course eventually K_{AS} and K_{BS} will be changed, at which time the cached keys will become invalid. Notice that there is no necessity to store the renewable tokens securely as long as they remain in their encrypted state and only decrypted at their time of use.

4. Discussion

4.1. Comparison with Previous Protocols

As already mentioned, the general structure of the new protocols has many similarities with well known protocols but the fundamentally different method to obtain freshness has far-reaching consequences. One notable protocol with similarities is due to Gong from 1989 [6]; in that protocol one of the principals obtains freshness by input while the other receives a nonce with the session key. It is also worthwhile to compare a recent protocol of Kao and Chow [9]. They employ a novel method to obtain freshness which involves using a short-term key just for this purpose. The following is a successful run of their protocol.

1. $A \rightarrow S : A, B, N_A$
2. $S \rightarrow B : \{A, B, N_A, K_{AB}, K_t\}_{K_{AS}}, \{A, B, N_A, K_{AB}, K_t\}_{K_{BS}}$
3. $B \rightarrow A : \{A, B, N_A, K_{AB}, K_t\}_{K_{AS}}, \{N_A, K_{AB}\}_{K_t}, N_B$
4. $A \rightarrow B : \{N_B, K_{AB}\}_{K_t}$

As above, $\{M\}_K$ denotes the message M encrypted with key K where (presumably) the encryption provides both message confidentiality and integrity. The short-term key K_t is used only for the duration of the protocol run and is then destroyed by both parties. This means that it is unreasonable to assume that the key K_t will be found by an attacker and so used in a replay attack. This enables the protocol also to lower Gong's bound for nonce-based protocols (again within a different security model). However, there are other security implications of their method; in particular user A is able to force use of an old key - a curious property not discussed by the authors. Furthermore, it does not obtain the flexibility of key caching or re-use since the nonce chosen by A is included in the reply from the server.

There is also a certain similarity between the methods advocated here and the 3 party EKE protocol described by Steiner, Tsudik and Waidner [13]³. In their 3-party EKE protocol principals A and B both contribute to the key value and thus obtain freshness by input to the key. A trusted server shares secret passwords with A and B and uses these to transfer authentication between them. An advantage of the 3-party EKE over the protocols presented above is that it provides *forward secrecy*; that is, compromise of secrets does not compromise previously agreed session keys. However, because both A and B need to contribute their values to the server before getting their reply, the number of rounds

³I am grateful to one of the anonymous referees for pointing out this similarity.

remains the same as for conventional uses of nonces. Also, because the server responses depend on the user input, key caching and re-use are not possible.

The comparison with 3-party EKE suggests the following novel protocol which may be viewed as a combination of 3-party EKE with the ideas presented in this paper. In the following g is the generator of an appropriate multiplicative group such as the integers modulo some large prime. The other notation is the same as elsewhere in the paper.

1. $A \rightarrow S : A, B$
2. $A \rightarrow B : A, g^{N_A}$
3. $S \rightarrow B : \{A, B, K_S\}_{K_{BS}}$
4. $S \rightarrow A : \{A, B, K_S\}_{K_{AS}}$
5. $B \rightarrow A : B, g^{N_B}$
6. $B \rightarrow A : [g^{N_A}]_{K_{AB}}$
7. $A \rightarrow B : [g^{N_B}]_{K_{AB}}$

Here the session key is calculated by A as $K_{AB} = (g^{N_B})^{N_A K_S}$ and by B as $K_{AB} = (g^{N_A})^{N_B K_S}$. This protocol has the advantages of the protocols examined earlier and also has the forward secrecy property of 3-party EKE. Notice that it is necessary here that the function $f(k, N_A, N_B) = g^{k N_A N_B}$ has the required security properties as described in section 2.1. Also it should be noted that one feature of 3-party EKE has been lost, namely that the shared keys could be passwords (indeed this was the whole motivation for the original EKE protocols [2]); in the protocol above the shared keys K_{AS} and K_{BS} must be full length keys not vulnerable to dictionary attacks

4.2. Vulnerability after Key Compromise

The security analysis given above makes the assumption that the key K_S has not been compromised. Since it is never used as a session key, or indeed to encrypt any information at all, this is a reasonable assumption in normal operation of the protocol. However, the possibility that at sometime a key K_S becomes compromised cannot be discounted. Now this may just as likely happen to a long term key such as K_{AS} in any protocol using shared keys, and the expected consequence would be that the key would be replaced. However, it was pointed out by Gong [8] that there is a significant difference here from protocols which gain freshness from an element received with the message.

Suppose, for example, that K_{AS} is compromised. Then A should ensure that it is replaced with a new value. Then any renewable tokens $\{K_S, A, B\}_{K_{AS}}$ previously used with any B are also compromised. However, the corresponding token $\{K_S, A, B\}_{K_{BS}}$ remains valid for B whether K_{AS} is

Explore Litigation Insights

Docket Alarm provides insights to develop a more informed litigation strategy and the peace of mind of knowing you're on top of things.

Real-Time Litigation Alerts



Keep your litigation team up-to-date with **real-time alerts** and advanced team management tools built for the enterprise, all while greatly reducing PACER spend.

Our comprehensive service means we can handle Federal, State, and Administrative courts across the country.

Advanced Docket Research



With over 230 million records, Docket Alarm's cloud-native docket research platform finds what other services can't. Coverage includes Federal, State, plus PTAB, TTAB, ITC and NLRB decisions, all in one place.

Identify arguments that have been successful in the past with full text, pinpoint searching. Link to case law cited within any court document via Fastcase.

Analytics At Your Fingertips



Learn what happened the last time a particular judge, opposing counsel or company faced cases similar to yours.

Advanced out-of-the-box PTAB and TTAB analytics are always at your fingertips.

API

Docket Alarm offers a powerful API (application programming interface) to developers that want to integrate case filings into their apps.

LAW FIRMS

Build custom dashboards for your attorneys and clients with live data direct from the court.

Automate many repetitive legal tasks like conflict checks, document management, and marketing.

FINANCIAL INSTITUTIONS

Litigation and bankruptcy checks for companies and debtors.

E-DISCOVERY AND LEGAL VENDORS

Sync your system to PACER to automate legal marketing.