FLAMeS: Fast, Loss-Tolerant Authentication of Multicast Streams

Bob Briscoe
<bob.briscoe@bt.com> <www.btexact.com/people/briscorj/>
BT Research, B54/130, Adastral Park, Martlesham Heath, Ipswich, IP5 3RE, England
Tel. +44 1473 645196

12 Mar 2000

Abstract

{TBA}

Keywords: Data communication, Networks, Internet, Group communication, Multicast, Broadcast, Security, Multimedia, Streaming, Authentication.

1 Introduction

{TBA}

2 Background and Requirements

{I will re-write this whole section, so don't bother reading it}

IP multicast deliberately designed on an open model where anyone can send to a multicast address and anyone can join to receive. Reception {. . . ?}

Authentication of a communication is usually achieved, at least initially, by some form of digital signing of the message. Where the parties have no previous relationship, asymmetric key cryptography is the most convenient way to authenticate messages. The signer, Alice, typically encrypts a digest of the message with her private key and sends this digital signature with the message. This is convenient because anyone can access a certified directory of public keys to check that the signature decrypts with the sender's public key back to a hash of the message. However, asymmetric key cryptography (e.g. RSA) is deliberately a computationally expensive operation — a dedicated 300MHz Pentium processor can typically only achieve under 80 signings or 2000 verifications per second for typical packet sizes [33]. This is before any time has been allocated for the most processor-hungry task of most streaming applications — servicing software codecs. Worse, often applications involve at least duplex streams, while layered codings or multi-media applications potentially introduce many more streams. And worse still, many mobile devices have considerably lower powered processors than this. Also note that Moore's Law does not save us here, because, as processors get faster, asymmetric key lengths will be increased to maintain equivalent strength. So where a stream of messages have to be authenticated (e.g. audio or video packets), faster message authentication is typically used.

{Not sure this next bit is the best order to introduce this by starting straight into technique rather than principle. . . }

A session key is randomly generated by the signer and passed to the intended recipient under the protection of her public key. Each message is then passed through a hash function initialised with this session key to create what is commonly termed a message authentication code (MAC). Each message is sent with its MAC to prove the sender's authenticity. This still leaves open the possibility of some eavesdropper, Eve, copying the message in transit and using the copy elsewhere inappropriately. Such a replay attack can be foiled by a time-stamp being added to the message before generating the MAC (a technique just as necessary when using public key cryptography).

{Distinguish non-repudiation of origin and of receipt and authentication of id of sender to receiver and to others}

The above MAC-based solution only works for one-to-one communication. It relies on the recipient, Bob, sharing a secret session key with the sender, Alice. Bob knows he didn't authenticate the message, so if he knows he also hasn't passed on the key, the only other person who could have must have been Alice. As far as Bob is concerned, if Alice is careless enough to pass the key on to someone else, she has to accept the consequences of what that third party might authenticate in her name. No-one considers MACs as proof to a third party, so Bob cannot fool anyone else that Alice sent a message she didn't. If Bob creates his own message, authenticates it with the session key and presents it to Alice, Alice knows she didn't authenticate it, so Bob can only fool himself.

If Alice tried to use a MAC to send an authenticated message to a *group* of people, she would have to reveal the secret session key to them all so that they could verify she sent it. But then any one of the recipients, say Carol, could fool any one of the others, say Bob, by sending messages with MACs based on Alice's session key. The MAC approach is only sufficient if the recipients are only concerned about ensuring *anyone* within the group is the sender.

When authenticating IP multicast streamed media such as audio or video, if a scheme doesn't tolerate a degree of packet loss it is effectively useless, as reliable repair of every packet to every receiver is very expensive and anyway deliberately unnecessary for real-time media codings. Although studies that have characterised losses on the experimental multicast backbone (MBone) don't claim to necessarily be typical, the traces used are not atypical either (e.g. the May 1996 NASA shuttle video multicast [15]). In this study, only some 15% of receivers experienced zero or very low losses with about 40% losing more than 20% of packets.

LSMA taxonomy of requirements [2]
EXPRESS [19]
Ballardie [3]
KHIP [31]
Canetti [8]
Wong/Lam [33]
Gennaro/Rohatgi [12] {Genaro buffers to authenticate before sending, whereas FLAMeS passes across the data as soon as it is ready, and the authentication arrives later.}
Anderson et al (Guy Fawkes protocol) [1]
Perrig [26]

Thus, to summarise, the state of the art offers no sender authentication technique for group communications with an overhead even approaching the equivalent of those used for two-party communications.

3 FLAMeS scheme

Below we introduce our stream authentication scheme, FLAMeS. We start with a brief description of the initialisation required. However, some of the detail of the initialisation gets in the way of the explanation so it is deferred — to keep the suspense to a minimum! Variants are described, each of which trades off a different requirement against the others, but full discussion of the applicability of each variant is also deferred until after they are all introduced.

The scheme can be used to authenticate message streams at any appropriate level, e.g. link frames, network packets or application data units. Therefore, we will use the neutral term 'message' for each unit of data that is to be authenticated. We model a stream as a sequence of messages from a single source:

- M_i is the ith message in the sequence, with i ascending with time.
- Message, M_i, consists of at least a data payload, D_i and a sender time-stamp, $C_{S,i}$.

3.1 Set-up

Before sending the first message, the sender must randomly generate H initial keys (H is one, two or more depending on the variant). As a concrete example, these keys should be 128b wide. From each of these the sender then creates a sequence of keys, each of which is blinded from the last (often termed a hash chain). Two such chains are shown in Fig 1a). The duration of the stream must be divided into time-slots with enough keys for one to be assigned to each. How to determine the time-slot duration will be discussed later. Note that none of the data for the messages needs to be known for the sequence of keys to be created, only how many keys are needed (the product of the expected session duration with the key frequency). However, if the session is open-ended, any practical chain length can be chosen, as chains can be chained together with a little more complexity. We discuss the practicalities of chain generation later, suffice to say that two key chains for an hour of audio consisting of 20ms packets with one packet per key time-slot can typically be generated in under a second on today's average PCs. We now define notation needed for set-up:

- $K_{h,j}$ is the jth key from the end of the hth key sequence, with j decrementing with each blinding.

- h is the index of each sequence of keys. For instance, if there are two key sequences ($H = 2$), h takes values 0 or 1.

- J is the index of the first key in each sequence, thus the initial randomly generated key in each sequence is $K_{h,J}$. The last key is arranged to be $K_{h,0}$ making $J + 1$ keys in each sequence.

- Each step of the chain is computed from the mapping $K_{h,j} = b(K_{h,(j+1)})$ where the function, $b(\cdot)$, is a well-known blinding function such as the MD5 hash [28] or the NIST secure hash [25]. That is, a computationally limited adversary cannot find a key from its blinded value or any further blinded values down the chain. Further, because the result of one blinding is used for the next, $K_{h,j} = b^{J-j}(K_{h,J})$. That is, the jth key is found by running the blinding function recursively $J - j$ times starting with the initial key.

Keys don't necessarily map to messages on a one to one basis, which is why we use i for the message index and j for the key index.

The keys at the ends of each chain, $K_{h,0}$, are signed, typically using the private key of the sender and sent to all receivers. For example, taking the scenario of an Internet real-time packet stream, a typical way to achieve this would be by including the end keys in the session description using the session description protocol (SDP) [17] (see later for detailed example). Session descriptions can either be announced using the session announcement protocol as a transport (SAP) [18, 16] or targeted at individuals by explicitly inviting them into the session with the session initiation protocol (SIP) [14]. Both these transports include the facility for authentication of the session description. A representation of such a signed session description is shown in Fig 1b), where A_0 is the digest of the session description including all the end keys, $k_{h,0}$ and where $s(A_0)$ is the signature of this digest using the sender's private key, K_{-s}.

In all the figures in this paper, each thick black arrow represents a one-way dependency of one value on another; thus $y \rightarrow x$ and $z \rightarrow x$ mean x depends on both y and z, neither of which can reasonably be deduced from x.

3.2 S/KEY

As our scheme builds on the seminal work of Lamport [23], later termed the S/KEY authentication scheme [13], we will first describe a simple scheme based purely on S/KEY, then explain its weaknesses in a group scenario. {check S/KEY is exactly like this}

For this scheme, only one key chain is required so $h = 0$ only. Also, unlike our own scheme, we don't require a time-stamp with the data in each message, but instead a simple sequence no., i, (Fig 2). In S/KEY, there is a one to one mapping between keys and messages, i.e. $i = j$. As the sender generates each message in the stream, M_i, it accompanies it with the corresponding key, $K_{0,j}$. Because the key chain has been indexed in descending order but j increases as the stream progresses, the key used for each successive message works towards the *start* of the key chain. Fig 2 shows message, M_i, and the following message, M_{i+1}, accompanied by their respective keys working back up the key-chain.

Let us first consider the situation on receiving message, M_i, in Fig 2, accompanied by key, $K_{0,j}$. If the highest index of any message previously received in the stream is $(i - d)$, the receiver repeatedly hashes the key just received d times, thus calculating $b^d(K_{0,j})$. d will usually be 1, but might be greater if there have been losses. The receiver checks that the result matches the key $K_{0,(j-d)}$ that arrived with the most highly indexed previous message. Fig 2 also shows the situation after the arrival of the next key, $K_{0,(j+1)}$, with $d = 1$ (no loss). Note that this design achieves inherent loss-tolerance, as the validity of any newly revealed key in the chain can be validated across any gap in the message stream as long as at least one previous message in the stream has been received. Conceptually, the first message in the stream is that given earlier — in the session description. This is why we recommend the session description is delivered by reliable transport, either through regular repetition (SAP) or over TCP (SIP).

We show next that S/KEY alone has security weaknesses for group communications, however, if the attacks described below are unlikely, it can be used with the following precautions to offer a degree of authentication. Messages arriving at a receiver with the same index but different data are a sign of an attempt to spoof the stream. Therefore, any message with a sequence number equal to a previous message ($d = 0$) is discarded, keeping only the first to arrive. It may also be desired to flag an alarm if a duplicate index is received (possibly triggering adaptation to our stricter scheme below). Any out of order messages with sequence numbers less than the previous latest message ($d < 0$) cannot be assumed to be authentic but may be used if the lack of authenticity of that particular message is unimportant to the application. Misordered messages that don't even have a key that fits into the key chain should definitely be discarded.

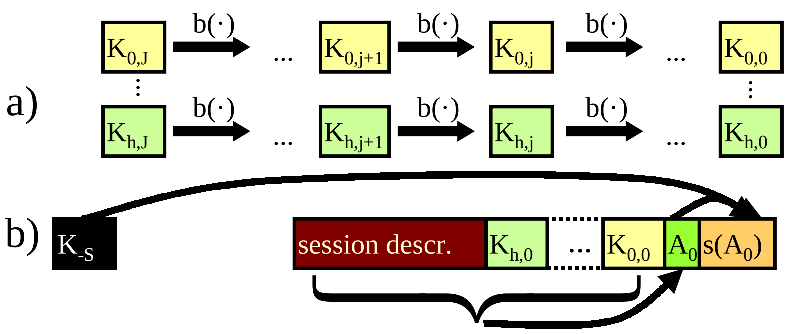

Figure 1: Hash chains of keys and authenticated initialisation information

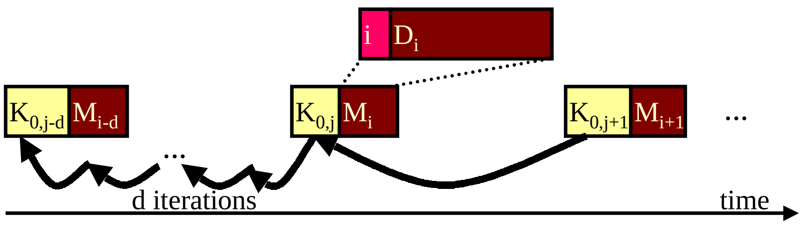

Figure 2: Immediate but network-dependent authentication

Unfortunately, as already noted, the security of S/KEY is reasonable, but not assured, without our further measures. Feasible ways for an attacker to spoof a message authenticated with S/KEY alone are:

- to receive the message, learn the next key backwards in the chain that is revealed with that message, then use that key to accompany a spoof replacement message and send these out in an attempt to overtake the first message (possibly by requesting differentiated latency from the network [9, 6] or by using a parallel network);
- to produce a spoof message as above, but send it out only to receivers where it is known that the original message was lost or delayed before it reached them (e.g. in response to a reliable multicast local repair request [11] or by taking advantage of misordered arrival of messages as already described);
- to conspire with a network provider to delay or remove the original message from part or all of the multicast tree and replace it with a spoof message created as above.

Thus S/KEY gives 'network-dependent' authentication — we use the term to refer to the set of issues above, being a more relevant term than 'man-in-the-middle' for the group communications context. All these attacks are remotely possible but nonetheless relatively hard to mount. Later we will discuss exactly how difficult these attacks really are, showing that this basic scheme is surprisingly strong against attackers without control of the network, given its simplicity and efficiency. Our subsequent scheme closes all these holes at the expense of marginally less simplicity and efficiency, but still a good deal better than the state of the art.

3.3 Immediate then certain authentication

Just like S/KEY, the FLAMeS scheme uses successively earlier keys back along a hash chain. For the first variant of this scheme, only one key chain is required so again $h = 0$ only. However, unlike before, as the sender generates each message in the stream, M_i, it uses the corresponding key, $K_{0,j}$, to initialise a message authentication code (MAC), typically using a keyed hash [32].

- $A_{h,i} = a(M_i, K_{h,j})$ is the message authentication code of M_i with respect to $K_{h,j}$, using a well-known MAC function $a(\cdot)$, such as HMAC [4, 21].

The sender sends each message in the stream accompanied by its MAC. However after *using* a key to create a MAC, the sender waits at least time $T_{h,G}$ before *revealing* the key. Thus the MAC accompanying each message is a commitment to a key in the chain, which can only be verified later when the key is revealed. As long as any message arrives before a deadline, which we show how to calculate later, each receiver can be certain that the key used to authenticate that message cannot have even entered the network, and therefore must still be secret. All the sender has to do is guarantee to withhold the key for the declared period, $T_{h,g}$.

We discuss the compromises around setting the value of $T_{h,G}$ later, but if it is set large enough, there is scope for more efficient use of the key chain. Specifically, because exposure of the key is delayed, the same key may be reused to create MACs for multiple messages, as long as its exposure is deferred until $T_{h,G}$ after its *last* use. All that is necessary is to use the simple, succinct technique given later to declare regular time-slots of duration $T_{h,K}$, within which a key may be re-used. At the other extreme, if the message frequency is variable, it is possible that some time-slots will not contain a message at all. However, as each time-slot passes, the key must still be systematically progressed back along the chain, to keep the correct keys in the correct time-slots.

Fig 3 shows how the key, $K_{0,n}$, that was first used to create the MAC, $A_{0,i}$, for message, M_i, is revealed more than $T_{0,K} + T_{0,G}$ later than the message it relates to. Just as $K_{0,n}$ is revealed with a later message, earlier key, $K_{0,j}$, is revealed when message M_i is sent, allowing an earlier message (not shown) to be verified. Also, the first time $K_{0,j}$ is revealed, it gives the same immediate but weaker authentication of M_i that S/KEY alone would give. Thus, although full authentication is delayed, immediate but slightly weaker authentication is also achieved.

In the example shown, the same key, $K_{0,n}$, is reused to generate MACs, $A_{0,i}$ and $A_{0,(i+1)}$, because their respective messages, M_i and $M_{(i+1)}$, are in the same time-slot, of duration $T_{0,K}$. As well as the same key being *used* throughout a time-slot, the keys *revealed* with every message in the same time-slot are all equal to each other. To save communication overhead, each key need not be revealed repeatedly within a time-slot, but doing so is the simplest way to be resilient to losses. Repeated exposures of the same key clearly loses the benefit of immediate S/KEY-like authentication after the first time. Thus, if immediate authentication is important, it is recommended that $T_{h,K}$ is set so that there is just one message per key time-slot. This ensures immediate, authentication of all messages, notwithstanding the fact it is weaker than the later full authentication.

We now describe how the sender works out the time-slots of the keys used and revealed in any message. Despite exposure of the key being delayed, there is no need to declare which key is in which message. This is defined implicitly by the time-stamp hint, $C_{S,i}$, in each message. For each message, the sender is committed to *using* the key with the index that is the number of whole key change intervals since the first key was used. The sender is similarly tied down as to which key to *reveal* in each message, by having to adhere to the 'guaranteed silence' period. Therefore, on reading $C_{S,i}$, any receiver can repeat the same calculations as the sender to find which key was used and which key has been revealed. Thus, as long as the sender declares the relevant constants to all receivers in the session description, they can both work out the full schedule of when any key should be used or revealed. Formally, the sender will use $K_{h,n}$ to authenticate message M_i, containing time-stamp $C_{S,i}$, where:

$$n = L_h + \lfloor (C_{S,i} - C_{S,h,L})/T_{h,K} \rfloor . \qquad (1)$$

In the same message, M_i, the sender will reveal an earlier key, K_j, where

$$j = L_h + \lfloor (C_{S,i} - C_{S,h,L} - T_{h,G})/T_{h,K} \rfloor . \qquad (2)$$

Where:

- $C_{S,h,L}$ was the sender clock reading when the first time-slot started;
- L_h was the key index of the first key to be *used*, $K_{h,L}$, in each chain. Typically $L_h = 1$ for all h (Fig 1), but allowing other values is necessary later (to cater for late join).

We now describe in outline the receiver's strategy for verifying each message, M_i:

1. Work out the time-slot of the key, $K_{0,n}$, *used* for the MAC in the newly arrived message (using its sender time-stamp, $C_{S,i}$ and equation (1);
2. Check the new message has arrived before the deadline when the sender may have revealed the key used for its MAC (the simple method is described later);
3. Work out the time-slot of the key, $K_{0,j}$, *revealed* in the newly arrived message (using its sender time-stamp, $C_{S,i}$ and equation (2));
4. Work out the difference, d, between the time-slot index of this newly revealed key and the previous latest time-slot of a revealed key;
5. Validate the newly revealed key by repeatedly hashing the previous latest key d times, as with the S/KEY scheme above, immediately giving probable, but not certain, authentication;

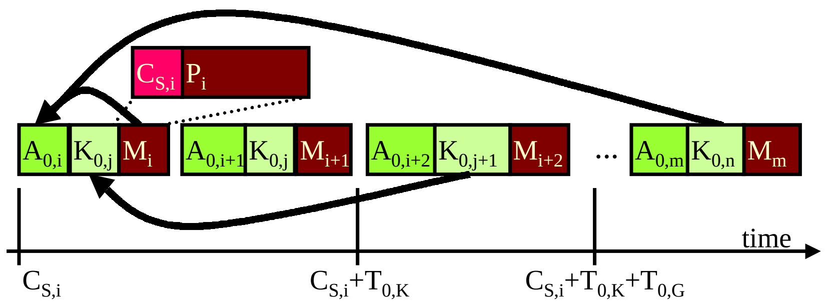
$C_{S,i}$ P_i
$A_{0,i}$ $K_{0,j}$ M_i
$A_{0,i+1}$ $K_{0,j}$ M_{i+1}
$A_{0,i+2}$ $K_{0,j+1}$ M_{i+2}
...
$A_{0,m}$ $K_{0,n}$ M_m
time
$C_{S,i}$
$C_{S,i}+T_{0,K}$
$C_{S,i}+T_{0,K}+T_{0,G}$

Figure 3: Immediate then certain authentication

6. Given the newly revealed key and having worked out the time-slots of the keys *used* in each earlier message (in previous runs of step 1) identify the message(s) that can now be validated, say M_g;

7. then use $K_{0,j}$ to validate the earlier MAC, $A_{0,g}$. That is, $a(M_g, K_{0,n}) = A_{0,g}$.

Because the sender has to commit to the MAC when the message is sent, this scheme is proof against a man-in-the-middle attack (by Mallory, say). Mallory cannot commit to a new MAC for a spoof message before a valid key has even entered the network. He has to delay the message until the valid key is revealed so that he can produce a valid MAC. A message that arrives at the receiver at a time when it is possible that the 'guaranteed silence' timer, $T_{h,G}$, has expired at the sender cannot be deemed authentic by that receiver. This is because the associated key cannot be guaranteed *not* to have entered the network. With this discipline in place, which is fully described later, Mallory has no scope for his attack. This is, in effect, a one-way version of the Interlock Protocol [27], using expiry of implicit timers instead of explicit replies in order to interlock commitments and their associated keys. Although the authentication given by FLAMeS is delayed, once it arrives it is as strong as the signature used to sign the initial session description with some minor caveats that we discuss later.

3.4 Wide variations in receiver delay

In large-scale multicast environments, it is quite possible for delivery delay to be very low for some receivers but awful for others. Rather than have to set $T_{h,G}$ to satisfy the lowest common denominator receiver, we now describe a way to allow 'closer' receivers to be certain of the source identity even if the same message doesn't arrive at more 'remote' receivers until after they know the sender might have revealed the key. This presupposes that the application needs authentication as quickly as possible, but still works the longer it is delayed. All that is necessary is for the sender to include two (or more) MACs with each message, and to reveal the key for each after progressively longer 'guaranteed silence' delays, $T_{h,G}$.

This is why we showed the general case in Fig 1, where there are H key chains, with h taking all possible values from 0 to $H-1$. For each value of h, the sender must declare the timing parameters $C_{S,h,L}$, L_h, $T_{h,K}$, $T_{h,G}$, although often the first two or even three might be the same for all h.

Fig 4 shows a typical example where H=2 (two MACs and two keys per message). For message, M_i, the key, $K_{0,n}$, that created the first MAC, $A_{0,i}$, is revealed soonest, while the other key, $K_{1,n}$, that created the other MAC, $A_{1,i}$, appears later. That is, $T_{1,G}$ is set longer than $T_{0,G}$. As soon as M_i appears at any receiver, the keys with it can be tested to check they hash to previously revealed keys. This immediately increases the probability that M_i is authentic. However, it may arrive at a 'close' receiver before the deadline when the receiver can be sure $K_{0,n}$ hasn't entered the network, but arrive at a more distant receiver after this deadline. As long as it arrives at the 'further' receiver before the deadline when it becomes possible that $K_{1,n}$ has entered the network, the further receiver will immediately know it is still worth waiting for the second key to arrive. When $K_{0,n}$ arrives at each receiver only the 'closer' one can be certain of the authenticity of M_i, while the 'further' receiver only experiences increased probability of certainty. When $K_{1,n}$ arrives, certain authenticity is assured for the 'further' receiver. The 'closer' receiver can ignore the second MAC and its later key.

Earlier, we discussed how only one key should be used per time-slot to avoid losing the immediate S/KEY-like authenticity test when the same key is revealed multiple times. When there is more than one key chain, this restriction need only apply to one of the chains. The longer 'key silence' delays might as well be accompanied by longer key change intervals to minimise the cost of high key change frequency. Fig 4 shows such a scenario, where K_0 changes every message, while K_1 changes every two messages.

If desired, it would be perfectly possible to use more than two MACs per message, with the key for each being progressively delayed. This would neatly cater for extreme heterogeneity of delivery latency either in space across a multicast tree or in time during a session. Messages that arrived with low latency could be authenticated by an early key exposure and those arriving with poorer latency could still be authenticated with certainty by a later key exposure. All receivers would continue to be able to take advantage of the immediate but weaker authentication afforded by the keys accompanying the messages, as already described.

3.5 Low frequency message streams

This is a variant of FLAMeS that merely caters for scenarios where messages are relatively infrequent compared to the 'guaranteed silence' timer. That is, for a significant proportion of messages, $(C_{S,i+1} - C_{S,i}) \gg (T_{h,K} + T_{h,G})$. In such cases, it makes sense to reveal the delayed key on its own, rather than waiting to attach it to the next message.

Fig 5 shows that, rather than $K_{0,n}$ appearing in a later message, it appears on its own as soon as the timer has expired that allows the sender to reveal it. In order for the receiver to know which key is which, the lone key is accompanied by a time-stamp hint which allows the key's index to be calculated from equation (2) above. If the scenario ensured that there was always one message per key at a regular frequency, a simpler alternative would be to refrain from sending any hint altogether, or to include just the key index itself, n, as the hint.

Revealing keys on their own in this way is also necessary to 'play out' the end of a session under the regular FLAMeS scheme. That is, once the data session has ended, it is still necessary to transmit the outstanding keys for a further time $T_{h,G}$ after the last message that contains a real payload.

4 Set-up issues

We now return to the details of the set-up phase as promised earlier.

4.1 Implicitly authenticated time-stamps

In FLAMeS, authentication of each message depends on the time at which the message was sent. However, it would be no good writing a time-stamp into the message then relying on authenticating the combination of time-stamp and message. An attacker could just re-write the time-stamp, exploiting the circular dependency of the time-stamp's own authentication on itself. Instead each key in the hash chain is assigned to a time-slot in advance. Thus, when one of the keys is used to produce the MAC of a message, it implicitly defines the time-slot in which the message was sent. To set up this arrangement (illustrated later in Fig 7) requires three extra steps:

The first step simply involves declaring a set of four time-related parameters in some session description (in fact, a set for each key chain index, h, in use). As specified in the key-chain set-up above, the session description is still signed with the sender's private key, ensuring that these four parameters are also fully authenticated (Fig 6):

- L_h, the index of the key to next be used, $K_{h,L}$. The aim is to synchronise around the time that it is planned to first use this key;
  - Prior to the session starting, L would always be 1 making its declaration redundant. This parameter only becomes useful for optimising late-join efficiency (see later);
- $C_{S,h,L}$, the clock reading at the sender when it is planned to first use key, $K_{h,L}$;
  - Again, the situation is considerably simpler if late-join is not to be catered for. $C_{S,h,1}$ would be the same for all h because all key chains would start at the same time. Typically the session start time might have already have been defined elsewhere making this parameter redundant. For example, if SDP were used to describe the session, it includes its own session start time field, '`t = <start time>`';
- $T_{h,K}$, the key change interval the sender intends to use;
- $T_{h,G}$, the 'guaranteed silence' period — the time interval after last *use* of a key during which the sender guarantees it will not *reveal* the key. It is chosen by the sender to ensure there is a high probability that the last message

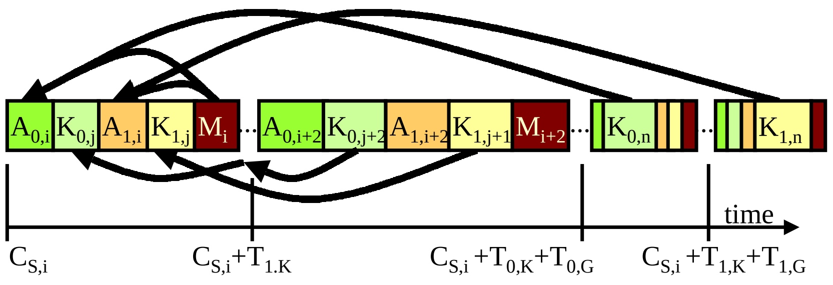

Figure 4: Immediate then certain then slower certain authentication

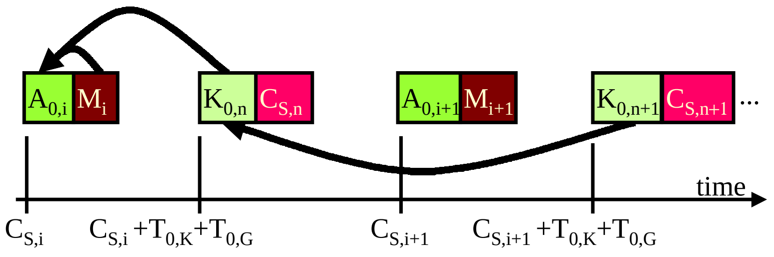

Figure 5: Low frequency message stream authentication

containing that key will have cleared the network for all receivers (potentially also allowing time for repair of losses). One could also think of $T_{h,G}$ as the 'assume-delivered' timer. Below, a simple technique is described for receivers to calculate with certainty the most pessimistic reading of their own clock at which this timer could have expired at the sender. However, clearly some messages might be delayed past the expiry of this timer or lost. The FLAMeS protocol simply states that the authenticity of such messages is suspect. This details of setting and measuring this timer are discussed in more detail later.

Note that, for clarity throughout this paper, we use C_S or C_R to represent clock readings at the sender or receiver respectively and T or t to represent time intervals between clock readings.

The second extra set-up step enables each receiver to be certain of the upper bound, t_D, on how much later the sender's clock could conceivably be reading than her own. The simple, common practice for establishing t_D is given in Appendix A. Essentially, the sender declares its clock reading and maximum clock drift rate in its authenticated reply to a single 'echo request' from each receiver (the terms sender and receiver are used in the sense of their eventual roles, not their temporary roles during this duplex message exchange). Note that we do not *synchronise* clocks, as neither party knows whether the other is more correct. We merely calculate the adjustment to add to the receiver's clock whenever it is needed. In fact, the pessimistic adjustment grows over time, so synchronising clocks just once would not be as accurate as calculating t_D on every occasion that it is needed. Later we discuss the implications of having to measure clock skew, as it is the only aspect of FLAMeS which requires a message in the 'receiver' to 'sender' direction.

The third addition is to ensure a time-stamp, $C_{S,i}$, is included in each message (Fig 3). This gives receivers a hint as to which key to use in order to verify a message, rather than having to blindly fish around for the correct one. A typical protocol that meets this criterion is the Internet real-time protocol (RTP) which is used for all sorts of real-time streamed media, such as audio or video [29]. The RTP header includes both a sequence number and a time-stamp. Note that authentication neither relies on the value of the sequence number in our use of S/KEY nor on the time-stamp in FLAMeS; both are merely hints. The true sequence number or time-stamp is the implicit one determined by the key used to authenticate the message.

Until now, we have loosely implied that the time-stamp hint in each message, $C_{S,i}$, is the time with respect to the sender's clock at the instant the message was sent. However, we have to take account of the practical issues of implementing an authentication 'layer' or 'module' in a communications stack. In fact, this time-stamp has to be calculated and

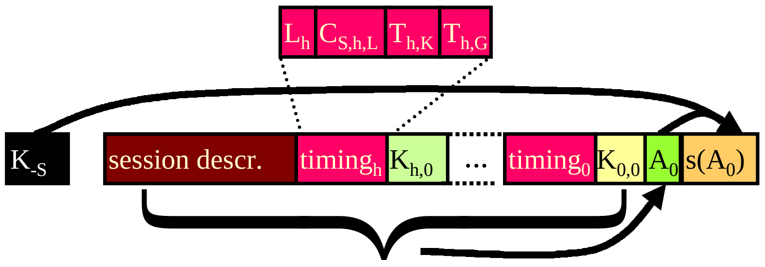

Figure 6: Timing declarations

included in the message payload even before the MAC is calculated. For instance, if the time-stamp is one already provided by the RTP protocol, it will certainly have been placed in the message during processing at a higher layer in the stack. Thus, the sender's authentication module doesn't refer to the system clock at all. All its logic is with respect to the time-stamps in the messages that are passed to it to authenticate. Thus, more precisely, the time-stamp hint in each message is taken as a given, rather than defined as the time of some generically significant event.

Fig 7 is a timing diagram with five related streams of events in the time dimension. The top three event streams are in the sender's scope and all use time relative to the sender's time-stamp in each message. In the receiver's scope covering the bottom half of the diagram, message reception in the upper event stream also considers time to be relative to the sender's time-stamp in each message. Only the bottom event stream uses time directly read from a system clock (the receiver's). For clarity, we use just one key chain with index, $h = 0$. To ensure a general discussion, we have chosen key change intervals, $T_{0,K}$, that are not integral multiples of the message data intervals. This might not be necessary in practice.

In Fig 7 we draw our messages in the 'sender (data)' stream with the start of each data unit 'box' coincident with its time-stamp. The sender describes the timings of the whole future session relative to $C_{S,0,1}$. Having set $T_{0,K}$ and $T_{0,G}$, the sender's authentication module has effectively constructed the time-slot plans for key use and key exposure over the entire session. These are shown in the event streams labelled 'sender (use)' and 'sender (reveal)' respectively. As messages are passed to the authentication module, it merely has to determine which key-use slot and which key-exposure slot the time-stamp falls in, then use or reveal the appropriate keys. For instance, for message M_{17}, it must use $K_{1,5}$ to produce the MAC and reveal $K_{1,2}$ with the message.

Similarly, each receiver can construct *the same* time-slot plan as the sender using the timing parameters in the session description. Then as messages arrive, each time-stamp hint enables the receiver to know which keys would have been used and revealed in which messages. Note that so far there is no direct reference to any system clock - the time-slot plan is created regardless of when messages arrive, or even whether they arrive in order. Thus, for instance, when M_7 arrives, the receiver can easily establish from the enclosed time-stamp hint that it used $K_{0,2}$ to create the MAC.

However, the receiver's authentication module must also construct a pessimistic plan of when to first reject each key used to create each MAC. For this she does use her system clock — in fact a worst-case increment on her clock based on a single clock comparison with the sender described later. For instance, at the arrival of M_7 (the end of the dotted arrow) she checks that her clock reading isn't after the deadline at which she should no longer trust messages that use $K_{0,2}$. In the case of M_{10}, she would quickly calculate that it used $K_{0,3}$, but then she would notice that it had arrived after her deadline for $K_{0,3}$. The authenticity of M_{10} would then be suspect.

It is recommended that the receiver authentication module should pass data to the application immediately, whether authenticated or not. In parallel it should advise the application on any new authenticity knowledge, whether for the current message or previous ones. As each message can be weakly authenticated immediately, then authenticated with certainty later, this arrangement allows the application to decide what action to take as a result of failure to authenticate any one message. This follows the principle of application layer framing (ALF) [10].

If a message is lost (e.g. M_{13} in the figure), authenticity verification is completely loss-tolerant, as already described.

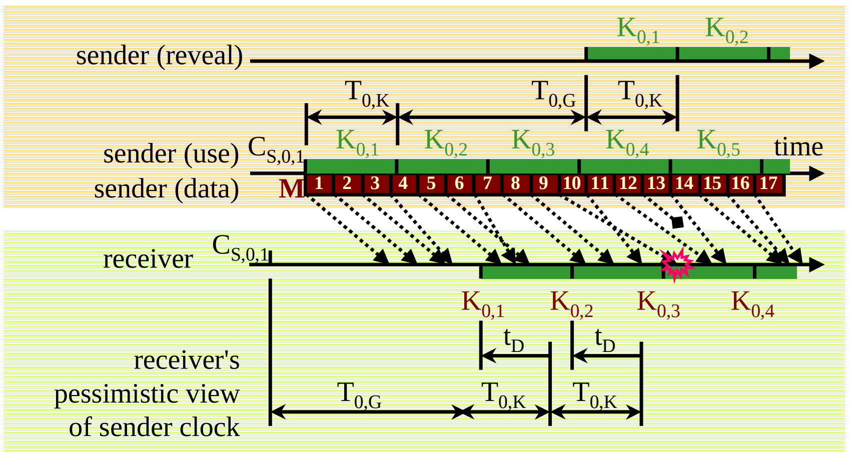
sender (reveal)
$K_{0,1}$ $K_{0,2}$
$T_{0,K}$ $T_{0,G}$ $T_{0,K}$
sender (use) $C_{S,0,1}$
$K_{0,1}$ $K_{0,2}$ $K_{0,3}$ $K_{0,4}$ $K_{0,5}$
time
sender (data) M 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17
receiver $C_{S,0,1}$
$K_{0,1}$ $K_{0,2}$ $K_{0,3}$ $K_{0,4}$
t_D t_D
receiver's pessimistic view of sender clock
$T_{0,G}$ $T_{0,K}$ $T_{0,K}$

Figure 7: Implicitly authenticated time-stamps

4.2 'Guaranteed silence' delay

The FLAMeS protocol relies on the sender not inserting a key, $K_{h,n}$, into the network until a message where the time-stamp is least the 'guaranteed silence' time, $T_{h,G}$, after the time-stamp of the last message to contain a MAC that used the key. Earlier we promised to discuss the issues relating to the setting of $T_{h,G}$. To set this timer, we have to be clear about the exact definitions of the events at the start and end of its duration and how these are measured by senders and receivers. Only then can we know what typical delays to include in our allowance for this timer.

The definition of the start of the guaranteed silence period from the sender's perspective depends on the definition of the time-stamp used in the particular scenario being designed for. For instance, if we use an RTP time-stamp, the RTP RFC defines it as: "...the sampling instant of the first octet in the RTP data packet. The sampling instant must be derived from a clock that increments monotonically and linearly in time..." [29]. From the sender's point of view, the end of the guaranteed silence period is also defined relative to the time-stamp in a later message. One might argue that the subsequent exposure of the associated key would also be subject to similar delays in the sender's stack. This line of argument might continue by asserting that, as long as all times were measured relative to the same type of event, delays would all cancel out, thus removing the need for any special allowance when revealing the MAC. However, when setting $T_{h,G}$, we have to take the pessimistic view that the message using a key might be subject to worst-case stack delays, while the later message revealing the key might happen to leave the sender with best-case stack delays. Thus, the difference between worst-case and best-case times required to buffer the whole message before passing it down the stack for authentication, let alone buffering before MAC calculation, cannot be ignored when we are deciding the value of $T_{h,G}$. In particular, if messages might be of different lengths, the maximum length message must be assumed when using a key, and the minimum length when revealing it later.

From the receiver's point of view, any definition of the FLAMeS protocol for a specific type of message will require a strict definition of the time that the message 'arrives'. Otherwise, for instance, some implementations might take this time as that of the arrival of the first octet while others might take it as the time when the whole message is buffered ready for MAC verification. We therefore recommend that the arrival time should be defined as that when the first sign of the message is presented to the receiver's authentication module. One might worry that, at the time when the start of the message appears for authentication, the end of the message might still be in transit on the network, and therefore still theoretically subject to spoofing attacks. This might be of concern if FLAMeS were being used for authentication at the application layer, where messages might be fragmented before sending over a network. However, it would not be of concern for packet authentication at the network layer in a store-and-forward network.

We have now established the list of events to include in the allowance for the 'guaranteed silence' timer. Assuming RTP time-stamps, the list starts with the sender application buffering the message before passing it down for authentication. The list ends at the receiver with the layer below authentication completing message buffering before passing it up for authentication. In between are all the other sources of delay in both stacks, as well as, of course, all the network delays.

However, this is only completes one side of the equation. We also have to consider which delays are included in the message that establishes clock skew (Appendix A). This is very simple. Because we are taking a pessimistic view, we assume the echo request message from data receiver to data sender might have had zero delay — in the absence of any other way of knowing how much stack and network delay there is in between. Thus, to minimise pessimism, it is recommended that the clock reading the sender declares in the reply is taken as soon as the echo request arrives, however long its reply takes to prepare for sending.

To summarise so far, the implication of this more exact discussion is that the 'guaranteed silence' delay will need to be set more pessimistically (longer) than might at first have been thought. We have also justified describing a message with a single sending or receiving time, when in reality the start and end of any message are separated in time. We allowed this by extending the 'guaranteed silence' delay to cater for the longest possible messages when using a key but the shortest possible when revealing it later. Thus, in Fig 7, the rest of the box after the start is just for display, as it doesn't actually matter whether data is sent until the next time-stamp or not, as long as it doesn't overlap the next message. Thus, although message M_4 is shown starting in the first key time-slot but ending in the second, all that is relevant is the time-slot in which the value of its time-stamp sits (the first).

All the above analysis is well and good, but the sender must somehow decide on a pragmatic value of $T_{h,G}$ to start a session. It can take one of two approaches. Either it just plans to stick to its decision (and receivers experiencing longer delay just have to accept it), or the sender adapts to delay conditions fed back, for example, in RTCP receiver reports [29]. In the former case, it will have to be more pessimistic about possible delay than in the latter, as there is no opportunity to change. In the special case where an unauthenticated session precedes an authenticated one, the sender can set the initial value based on receiver reports during the earlier session. However, the above analysis shows that delay values taken from in-session measurements must still be treated with caution before setting the guaranteed silence timer adaptively. For instance:

- if the sender knows that all messages sent so far have been of fairly invariant length but that future messages might exhibit greater length variation, it can add an adjustment to the worst-case delay measured so far to cater for its future plans;
- if the sender knows that receiver delay reports are measured at different points in the stack to those relevant for setting $T_{h,G}$ it can make a reasonable allowance this fact.

If adaptation requires the timer to change, the new value must be included in an updated session description and reported to receivers. There is no security issue if it is increased, but if it is decreased, the sender must be sure everyone has received the updated description before effecting the change. This depends on the reliability mechanism of the session description transport. If this is tightly coupled (e.g. TCP) there is no issue. If a more loosely coupled soft-state session announcement is being used, it will be necessary to announce the change well in advance to allow a number of repetitions before the change is effected and to ensure earlier session descriptions have expired in case all updates are missed.

The initial value of the timer cannot be predicted in general as it depends on the geographical scope of the particular session. However, considerable measurement data is available that characterises typical end-to-end delay in many parts of the Internet [7], which may help come to an initial decision.

4.3 Clock skew management

The single clock comparison during set-up is the only aspect of FLAMeS that requires a message in the 'receiver' to 'sender' direction. Unidirectional links (e.g. satellite) would have to use the back channel for this echo request. Although this is unfortunate, it is not disastrous as it need not occur every session. In fact, one calculation of this upper bound can theoretically last 'for ever' (in practice, 'for ever' might mean until either party's real-time clock batteries go flat). However, t_D represents pairwise state that each receiver needs to hold about each sender and therefore must be cached properly if it is to be held 'for ever'. If it is held for arbitrarily long times between sessions, it might, for instance, be held as a time-expired field that is indexed against each sender in a 'cookie' [22]. Despite the cookie state management mechanism originally being applied to Web sessions, it is designed to be generalisable to protocols based on HTTP, such as the real-time streaming protocol (RTSP) [30]. Alternatively, for sessions with many senders, as pairwise synchronisations mount up with each new sender in the session, it may be more efficient for each receiver to simply hold the maximum t_D for any sender. The idea would be to merely hold this for the duration of the session, then discard it. This assumes the variance of all values of t_D is small compared to $T_{h,G}$, which will only be the case if all the senders are reasonably synchronised to a standard time-source.

The FLAMeS protocol must also be arranged to ensure a large number of receivers do not overwhelm the sender with an implosion of simultaneous echo requests. Holding over clock skew state between sessions should in itself mitigate such effects, however further measures to spread the load might be necessary. The simplest technique is for receivers to trigger their echo request after expiry of a timer set to a random proportion of the interval between learning of the impending session and its start time.

4.4 Late Join

For very long sessions, late joining receivers would be confronted with the task of repeatedly hashing the key in the first packet known to them, possibly many millions of times per key chain, in order to verify it against the key(s) right at the end of each key-chain, $K_{h,0}$. For example, joining an audio session an hour late might require a second's worth of dedicated processing to catch up on authentication. By this time another fifty packets might have arrived. Although an extra second is unlikely to be important when the receiver is already an hour late, a simple optimisation is possible. The session description can be regularly updated to include the latest key already revealed, rather than the very first key. Of course, it would have to be signed each time it was sent. This applies whether participants are invited into the session (e.g. with SIP) or the session is repeatedly announced (e.g. with SAP). The earlier descriptions of the time-related parameters in the session description were deliberately generalised for late join. That is, as the session proceeds, it is possible to continually update the index of the next key to be revealed, L_h, and the clock-reading, $C_{S,h,L}$, scheduled for its exposure, rather than always having to relate back to the very first key to be revealed in the session.

4.5 Chained chains

For very long, possibly interminable, sessions with short key time slots, it might become impractical for the sender to generate the very last key to be used and hash it repeatedly right through to the very first. Instead, the key chain can be started from a practical intermediate point, then during the session, as this intermediate end of the session approaches, another intermediate point can be set from which to start another key chain. There are two ways to achieve this:

- the second key chain can be included with the first in an updated, signed session description (cf. late join above);
- the end key from the next chain can be included in the last few messages authenticated by the previous key chain, before switching to using the new key chain as soon as the old chain runs out (this would require a way to signal when to switch).

4.6 Deployment with existing protocols

As we have already said, FLAMeS authentication can be applied at any appropriate layer; link, network or application. This specification describes the general scheme, but two further levels of specification are required before it can be used for a particular session.

- A specific 'profile' specification is needed to define sizes of protocol fields such as MACs etc. and in order to apply the scheme to a particular scenario. For instance, to define the exact protocol headers when authenticating RTP messages.
- Declaration of the fact that FLAMeS was to be used to authenticate a particular session and of the FLAMeS parameters to use in that session.

Below we describe a typical illustration of the latter, showing how to declare the intention to use FLAMeS to authenticate Internet RTP media streams. This would also require a FLAMeS profile for RTP over UDP over IP multicast, but we don't intend to give those details here. The proper protocol for defining the authentication property of a media stream should be the session description protocol (SDP) [17]. In fact, the RTP specification specifically expects authentication to be achieved by adding headers to its messages before handing them down to the socket for encapsulation in the UDP transport protocol headers (or stripped the headers in the reception direction). However, no specific facility to name the media authentication scheme was included in the SDP specification at the time it was written, nor any way to give it parameters, other than by using the general purpose extensibility of SDP (there was, however, a facility to specify encryption keys). Thus, for experimental purposes, we will have to recommend use of the `<attribute>:<value>` argument to the SDP attribute type ('`a=<zero or more media attribute lines>`') in each media description section. Note that more than one media stream cannot share the same key chain. To incorporate all the set-up parameters, we suggest the following syntax:

```
v=2 o=Butler 2890844526 2890842807 \
  IN IP4 vserv.buck-pal.gov.uk
s=King's abdication speech
c=IN IP4 224.2.17.12/127
t=2873397496 2873404696
...
a=auth-type:FLA
a=FLA-H:1
a=FLA-TK:1000
a=FLA-TG:2500
a=FLA-C:2873397496000
a=recvonly
m=audio 49170 RTP/AVP 0
a=FLA-L:2000
m=video 51372 RTP/AVP 31
a=FLA-L:1000
...
```

We have invented all the attribute types starting with '$FLA-$', the meaning of each being obvious with respect to the notation used in this paper. The time parameters are all in milliseconds. Note in this case, the session start time, t, (in seconds) is redundant, being one thousandth of FLA-C. Note that any parameters in the session section of the description (before any '$m =$' lines) apply to all the following media streams. We have also had to invent the '$auth - type$' attribute. However, ideally, the SDP RFC should be re-issued with the addition of a type to define the authentication scheme. As type values are case-sensitive, we suggest using '$A =$', as in the following suggested improvement over the above example:

```
v=2 o=Butler 2890844526 2890842807 \
  IN IP4 vserv.buck-pal.gov.uk
s=King's abdication speech
c=IN IP4 224.2.17.12/127
t=2873397496 2873404696
...
A=FLA
A=H:1
A=TK:1000
A=TG:2500
A=C:2873397496000
a=recvonly
m=audio 49170 RTP/AVP 0
A=L:2000
m=video 51372 RTP/AVP 31
A=L:1000
...
```

The session description itself would be signed using the capabilities already provided in the transport protocols for SDP, as already described.

5 Performance and Security Properties

5.1 Security

We shall now attempt the rather demanding task of quantifying the strength of an authentication scheme which starts off relatively weak but gives strong authentication after a known delay. Firstly, we shall concentrate on the simpler task of quantifying the eventual authentication strength after the 'guaranteed silence' time. We shall assume the H=1 variant, but higher values of H (to cater for a wider heterogeneity of receiver delay) don't change the eventual authentication strength for any one receiver, as long as one of the delay values chosen is long enough for that receiver.

5.2 Eventual strength

As for any digital signature, the signature, s, of the session description depends on the strength of the one-way function, $b(\cdot)$, that creates the digest, A_0, of what is being signed, as much as on the strength of the private key. In turn, we have arranged this digest to be dependent on each successive key in the key chain. The only difference is that the digest depends on each key in the chain through successive iterations of the one-way function, rather than just one. Thus, assuming iterative operation of $b(\cdot)$ makes it no less easy to reverse than a single operation, s signs each key, $K_{0,j}$, with equivalent strength to a direct signature:

$$\begin{aligned} s &= s(K_{-s}, A_0) \\ &= s\big(K_{-s}, b(\texttt{<sess.descr.>}, \texttt{<timing>}_0, K_{0,0})\big) \\ &= s\big(K_{-s}, b(\texttt{<sess.descr.>}, \texttt{<timing>}_0, b^j(K_{0,j}))\big) \end{aligned}$$

Each message authentication code, $A_{0,i}$, also depends on each same key, $K_{0,j}$, through another one-way function, $a(\cdot)$:

$$A_{0,i} = a(M_i, K_{0,j})$$

Thus, the MAC of each message commits to a value known only to the sender that, when revealed later, can be verified as signed by the sender. We assume any receiver can be certain that the sender has not revealed $K_{0,j}$ to the network up to a known deadline as already described. Therefore, given our assumptions, the strength of the delayed authentication for multiple receivers is at least as good as that of the equivalent traditional authentication scheme, if it is based on the same MAC function and same size key. For instance, every FLAMeS receiver eventually has equivalent strength authentication to that of the MAC-based authentication used in IPsec [20] {Change this to the IPsec AH RFC} that is used for immediate authentication by a single receiver. In fact, the eventual authentication strength of FLAMeS is possibly stronger, as the key remains unrevealed during authentication rather than being shared via a message encrypted under the sender's private key as in IPsec.

5.3 Immediate strength

{Got to here — no point reading further}

{stuff here about network attacks — diffserv, overtaking, delaying, removing, RPF multicast routing being based on hop counts not minimising latency etc}

The mere presence of the delayed certainty also deters attackers from attempting a 'brief spoof'. The only motivation here would be to create regular short-term confusion as a form of denial of service. Given we will show that such attacks are already difficult to mount, the return wouldn't seem to justify the effort in the majority of realistic scenarios.

5.4 General precautions

As with any security system, a weak implementation of initialisation procedures can compromise the whole scheme. Possible vulnerabilities that aren't particularly specific to the proposal at hand are:

- to break the one-way blinding function in order to calculate keys earlier in the chain and use them to send spoof messages before the sender (in the time between messages or since the session initiation message)
- to predict the initial keys generated by the sender due to poor random number generation or to otherwise compromise the sender's system by various nefarious methods.

5.5 Storage and Processing Issues

{How to trade off storage against processing when producing the chain}

5.6 Efficiency

{Figures on:
number of one-way functions for authentication and verification per message;
message overhead of authentication.}

The authentication data required for this scheme is considerably smaller than other state of the art schemes. If, however, the overhead is still too much, the MACs can be truncated. For instance, the 'immediate' MAC, $A_{0,i}$, is only a short-term indicator of likely authenticity, therefore half its length or more could be omitted without serious implications. If only w_1 bits of $A_{0,i}$ were included, a similar number, say w_2 bits of $A_{1,i}$ could be omitted. Fig 8 shows the formation of the truncated MACs, $a'_{0,j}$ and $a'_{1,j}$. {Add bit about being the same or a bit longer} As long as the second MAC arrives within the 'guaranteed silence' period, this leaves the combined pair of MACs hardly any weaker than for a single MAC of equivalent bit width. This is because the combined probability of two invalid truncated MACs colliding simultaneously with two valid truncated MACs is the same as for one whole MAC of the same aggregate width. The overall MAC is slightly weaker as the authenticity of the remnant $a'_{0,j}$ is still 'network dependent' (defined earlier). Taking this approach, the bandwidth overhead for isochronous streams is one MAC (bit width w_a {change} in Fig 8) and two keys per message. {Add bit about being the same or a bit longer} Obviously, the protocol would have to define w_1 and w_2, so that receivers could repeat the same truncation before attempting a match.

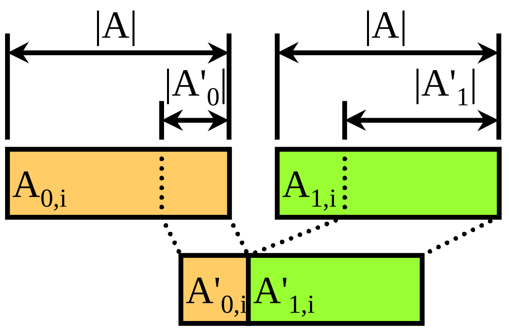

Figure 8: Two MACs for the price of one

{Stuff on equivalent probability of collision}

5.7 Loss-tolerance

{Discuss why FLA1 & 2 are so inherently loss-tolerant — i) if a key in a key chain is lost, as soon as a higher one is revealed it allows all the missed keys to be re-generated. ii) In FLA1 a message only relies on itself other than the key iii) In FLA2 the MAC and message always arrive together, making it impossible to lose one and not the other. The key arrives separately but lost keys can be regenerated as in i).}

6 Applications

This scheme would not be appropriate for an application where incoming messages were acted on immediately (e.g. purchases triggered by authenticated prices), unless roll back were easy if a message turned out to be spoof. However, we believe many applications will find delayed authentication quite adequate, especially as the 'guaranteed silence' timer is typically only a second or so.

Incidentally, this can be used as the basis of an extremely lightweight, authenticated multicast time-source, possibly replacing authenticated NTP [24].

7 Limitations and Further Work

8 Conclusion

Acknowledgements

References

[1] Ross J. Anderson, Francesco Bergadano, Bruno Crispo, Jong-Hyeon Lee, Charalampos Manifavas, and Roger Needham. A new family of authentication protocols. *Operating Systems Review*, 32(4):9–20, October 1998.

[2] Pete Bagnall, Bob Briscoe, and Alan Poppitt. Taxonomy of communication requirements for large-scale multicast applications. Request for comments 2729, Internet Engineering Task Force, URL: rfc2729.txt, December 1999.

[3] Tony Ballardie. Scalable multicast key distribution. Request for comments 1949, Internet Engineering Task Force, URL: rfc1949.txt, May 1996.

[4] Mihir Bellare, Ran Canetti, and Hugo Krawczyk. Message authentication using hash functions — The HMAC construction. *RSA Laboratories CryptoBytes*, 2(1), Spring 1996.

[5] Kenneth P. Birman. *Building Secure and Reliable Network Applications.* Manning, URL: `http://www.manning.com/Birman/index.html`, 1996.

[6] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss. An architecture for differentiated services. Request for comments 2475, Internet Engineering Task Force, URL: rfc2475.txt, December 1998.

[7] J. Bolot. End-to-end packet delay and loss in the Internet. *Proc. ACM SIGCOMM'93, Computer Communication Review*, 23(4), September 1993.

[8] Ran Canetti, Juan Garay, Gene Itkis, Daniele Micciancio, Moni Naor, and Benny Pinkas. Multicast security: A taxonomy and efficient constructions. In *Proc. IEEE Conference on Computer Communications (Infocom'99)*, volume 2, pages 708–716, URL: `http://www.wisdom.weizmann.ac.il/~bennyp/PAPERS/infocom.ps`, March 1999.

[9] David D. Clark. A model for cost allocation and pricing in the Internet. In *Proc. MIT Workshop on Internet Economics*, URL: `http://www.press.umich.edu/jep/works/ClarkModel.html`, March 1995.

[10] David D. Clark and David L. Tennenhouse. Architectural considerations for a new generation of protocols. *Proc. ACM SIGCOMM'90, Computer Communication Review*, 20(4):200–208, September 1990.

[11] Sally Floyd, Van Jacobson, Ching-Gung Liu, Steven McCanne, and Lixia Zhang. A reliable multicast framework for light-weight sessions and application level framing. *Proc. ACM SIGCOMM'95, Computer Communication Review*, 25(4):342–356, October 1995.

[12] Rosario Gennaro and Pankaj Rohatgi. How to sign digital streams. In *Proc. Advances in Cryptology - CRYPTO'97*, volume 1294, pages 180–197. Springer LNCS, 1997.

[13] N. Haller. The S/KEY one-time password system. Request for comments 1760, Internet Engineering Task Force, URL: rfc1760.txt, February 1995.

[14] M. Handley, H. Schulzrinne, E. Schooler, and J. Rosenberg. SIP: Session initiation protocol. Request for comments 2543, Internet Engineering Task Force, URL: rfc2543.txt, March 1999.

[15] Mark Handley. An examination of MBone performance. Research report ISI/RR-97-450, USC/ISI, URL: `http://www.aciri.org/mjh/mbone.ps`, 1997.

[16] Mark Handley. *On Scalable Internet Multimedia Conferencing Systems.* PhD thesis, Dept. of Computer Science, UC London, URL: `http://www.aciri.org/mjh/thesis.ps.gz`, November 1997.

[17] Mark Handley and Van Jacobsen. SDP: Session description protocol. Request for comments 2327, Internet Engineering Task Force, URL: rfc2327.txt, March 1998.

[18] Mark Handley, Colin Perkins, and Edmund Whelan. Session announcement protocol. Request for comments 2974, Internet Engineering Task Force, URL: rfc2974.txt, October 2000.

[19] Hugh W. Holbrook and David R. Cheriton. IP multicast channels: EXPRESS support for large-scale single-source applications. *Proc. ACM SIGCOMM'99, Computer Communication Review*, 29(4), September 1999.

[20] Stephen Kent and Randall Atkinson. Security architecture for the Internet protocol. Request for comments 2401, Internet Engineering Task Force, URL: rfc2401.txt, November 1998.

[21] H. Krawczyk, M. Bellare, and R. Canetti. HMAC: Keyed-hashing for message authentication. Request for comments 2104, Internet Engineering Task Force, URL: rfc2104.txt, February 1997.

[22] D. Kristol and L. Montulli. HTTP state management mechanism. Request for comments 2109, Internet Engineering Task Force, URL: rfc2109.txt, February 1997.

[23] Leslie Lamport. Password authentication with insecure communication. *Communications of the ACM*, 24(11):770–772, November 1981.

[24] David Mills, T. S. Glassey, and Michael McNeil. Authentication scheme extensions to NTP. Internet draft, Internet Engineering Task Force, URL: draft-mills-ntp-auth-coexist-01.txt See IETF stime working group charter for status: URL: `http://www.ietf.org/html.charters/stime-charter.html`, September 1998. (Work in progress) (expired).

[25] Secure hash standard. FIPS publication 180-1, NIST, U.S. Department of Commerce, Washington, D.C., April 1995.

[26] Adrian Perrig, Dawn Song, Doug Tygar, and Ran Canetti. Efficient authentication and signature of multicast streams over lossy channels. In *Proc. IEEE Symposium on Security and Privacy 2000*, URL: `http://paris.cs.berkeley.edu/~perrig/`, May 2000.

[27] Ronald A. Rivest and A. Shamir. How to expose an eavesdropper. *Communications of the ACM*, 27(4):393–395, April 1984.

[28] Ronald L. Rivest. The MD5 message-digest algorithm. Request for comments 1321, Internet Engineering Task Force, URL: rfc1321.txt, 1992.

[29] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. RTP: A transport protocol for real-time applications. Request for comments 1889, Internet Engineering Task Force, URL: rfc1889.txt, January 1996.

[30] H. Schulzrinne, A. Rao, and R. Lanphier. Real time streaming protocol (RTSP). Request for comments 2326, Internet Engineering Task Force, URL: rfc2326.txt, April 1998.

[31] Clay Shields and J. J. Garcia-Luna-Aceves. KHIP — A scalable protocol for secure multicast routing. *Proc. ACM SIGCOMM'99, Computer Communication Review*, 29(4), September 1999.

[32] G. Tsudik. Message authentication with one-way hash functions. *ACM SIGCOMM Computer Communication Review*, 22(5):29–38, October 1992.

[33] Chung Kei Wong and Simon S. Lam. Digital signatures for flows and multicasts. In *Proc. ICNP'98*, URL: `http://www.cs.utexas.edu/users/UTCS/techreports/index/html/Abstracts.19\%98.html\#TR-98-15`, 1998.

A Bounded clock skew

There follows a simple technique to establish the upper bound, t_D, on how much later the sender's clock might be reading than the receiver's [5, Ch.20]. We assume that both the sender and receiver know their maximum linear clock drift rates, C'_S and C'_R respectively.

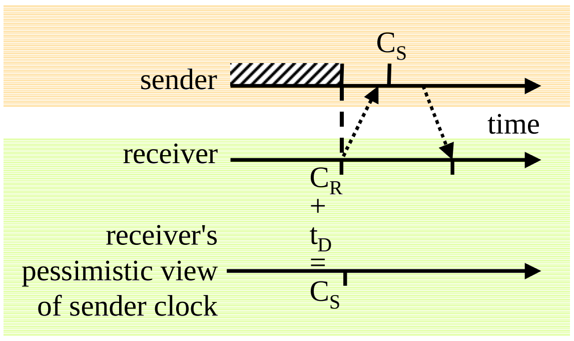

Figure 9: Bounded clock skew

The 'receiver' sends a ping-like echo request to the 'sender' at time C_R, which contains a nonce, N (again, the terms 'sender' and 'receiver' refer to their eventual roles, not their roles during this exchange). The sender replies with a signed message containing it's own clock reading, it's maximum clock drift rate and confirmation of the receiver's nonce, $\langle C_S, C'_S, N, S(A_S)\rangle$. $S(A_S)$ is the digest of the message signed with the sender's private key. This exchange is illustrated in Fig 9. As long as the response arrives (it is irrelevant when) the receiver only need consider the most pessimistic scenario. The sender's clock cannot have read C_S before the request was sent (the shaded region) as it would have also had to read C_S some time after the request was sent. Therefore the receiver knows the latest the sender's clock could have read when hers was reading C_R, making the maximum adjustment $(C_S - C_R)$. However, the more time that passes following the echo exchange, the more the two clocks might drift. The worst scenario is that the receiver's drifts backwards and the sender's forwards, both by their maximum possible drift rates. Thus, the receiver must assume that, some time, t_R, later than the echo request at time C_R, the latest the sender's clock could read is:

$$t_D = C_S - C_R + (C'_S + C'_R)t_R.$$

We now justify the use of an unauthenticated nonce in the 'receiver's' echo request. This request without the nonce need carry no information, other than the fact that it is an echo request. We need not worry about an attacker delaying the request as this merely makes the pessimistic estimate of the 'sender's' clock setting more pessimistic. However, it must not be possible for an attacker to create a spoof request just before the genuine one, then destroy the real one. Placing a nonce in the request allows the 'receiver' to satisfy herself that the earliest authenticated reply is in response to her own request, rather than an earlier spoof.

{We now discuss alternatives to this one-to-one exchange where the scenario is outside these limitations (impossible?).

One approach is for the sender to set $T_{h,G}$ much greater than typical round trip times and for the sender to declare the tolerance on its clock accuracy relative to an internationally recognised signal []. Then C'_i can be taken to be C_r plus a bit.?}

Document history

Version	Date	Author	Details of change
Draft A	14 Sep 1999	Bob Briscoe	First coherent draft for review by Security Futures, based on write-up in notebook 25 Jul 1999 ...based on original presentation of 24 Jun 1999 ...based on full ideas 30 May 1999 ...based on reading problem description in draft-canetti-secure-multicast-taxonomy-00.txt
Draft B	27 Sep 1999	Bob Briscoe	Amended to removed circular dependency on time-stamp using implicit timestamps from timeslot plan - from idea had after diffchar security workshop 14 Sep 1999, written up in notebook 28 Sep 1999
Draft C	21 Jan 2000	Bob Briscoe	
Draft D	21 Jan 2000	Bob Briscoe	Draft frozen to sent to Adrian Perrig (met 3 Nov 1999 and discovered had both had exactly same ideas in parallel, except I hadn't thought of multiple messages in one timeslot). Sent with covering note apologising for redundant variants that I'd realised were bogus.
Draft E	26 Jan 2000	Bob Briscoe	Frozen to start re-working to remove bogus variants, in order to become an RFC.
Draft F	08 Mar 2000	Bob Briscoe	Frozen to send to Perrig, Song, Canetti, and Tygar.
Draft F+	12 Mar 2000	Bob Briscoe	Continuing to write security & performance section...