Nark: Receiver-based Multicast Non-repudiation and Key Management

Bob Briscoe & Ian Fairman

Abstract: The goal of this work is to separately control individual secure sessions between unlimited pairs of multicast receivers and senders while preserving the scalability of receiver initiated Internet multicast for the data transfer itself. Unlike other secure multicast solutions, there are absolutely no side-effects on other receivers when a single receiver joins or leaves a session. Each individual receiver can also reliably prove whether any fragment of the data hasn't been delivered or wasn't delivered on time (e.g. late video frames). Further, each receiver's data can be subject to an individual, watermarked audit trail. The cost per receiver-session is typically just one set-up message exchange with a key manager. Key managers can be replicated without limit because they are only loosely coupled to the senders who can remain oblivious to members being added or removed. The solution requires a tamper-resistant processor such as a smartcard at each receiver. However, generic cards supplied by a trusted third party are used rather than cards specific to each information provider. The technique can be applied to other bulk data distribution channels instead of multicast, such as DVD.

1. Introduction

This paper explores techniques to maintain an individual security relationship between multicast senders and each receiver without compromising the efficiency and scalability of IP multicast's data distribution. We focus on issues that are foremost if the multicast information is being sold commercially. Of prime concern is how to individually restrict each receiver to extract only the data for which it has paid. Secondly, commercial information delivery systems should preferably include the capability for individual proof of delivery. Where both non-repudiation and transport reliability aren't intrinsic to the delivery system, the cost of providing customer support to handle billing complaints is likely to overshadow all other costs. However, where streamed information is concerned, simple proof of reception is not enough. Timely reception must also be provable. Thirdly, of particular concern with multicast information products is prevention or at least detection of unlicensed re-distribution of received information.

We adopt an approach where the key used to encrypt sent data is systematically changed for each new unit of application data. The keys are taken from a pseudo-random sequence seeded with a value initially known only to the senders. When a receiver wishes to join, it requires a smartcard or other securely encapsulated processor containing its own private key certified by a party trusted by both the receiver and the senders. At the end of each receiver's set-up phase, its card is running a key generator seeded with the same value as that of the senders and it contains a policy defining which keys the receiver is entitled to. The smartcard does no decryption; it merely hands out a key whenever a request conforms to the policy. The smartcard can record a summary of which keys it has given out that can be used as a non-repudiable 'delivery note' in the case of disputes over delivery.

Thus, whenever a receiver is added or removed, there is zero side-effect on other receivers. A special group key change doesn't have to be initiated because systematic changes occur all the time anyway. No keys need sending over the multicast, therefore reliable multicast isn't required. If key managers are delegated to handle requests to set-up receiver sessions, the senders can be completely oblivious of any receiver addition or removal. As long as the sender continues to change keys systematically, it doesn't need any control messaging from the key managers. Thus, there is absolutely no coupling back to the senders. The scheme is particularly suited to a receiver pre-payment model. In this scenario, any amount of key manager replication can be introduced. The key managers just give out session seeds and policies in return for pre-payments. Because no credit is given, each key manager stateless. Thus performance is linear with key manager replication and system resilience is independent of key manager resilience. In these respects, the model where all session state is stored at the receiver is similar to the Netscape cookie mechanism [IETF_RFC2109].

Thus we focus on a pragmatic scenario where evictions from the multicast group are typically planned at session set-up, but still might occur at arbitrary times. This avoids the huge overhead seen in other schemes to support regular unplanned evictions. Nonetheless, we do cater for the occasional unplanned eviction, although the scheme doesn't scale if the level of its use becomes high. Our thesis is that there are many applications that only rarely require premature eviction from a secure session. For instance, the business model for many pay-TV or pay-per-view services works on the basis of a pre-payment giving the right to a certain viewing window. Alternatively an account relationship is created and it is only necessary to evict an account holder if it becomes clear the account is in default.

By focussing on this pragmatic scenario, our scheme typically requires just one set-up message per receiver session, which can be with a highly replicated key manager rather than the sender. This exchange might, for instance, be a check that the receiver's identity entitles it to listen to information up to a certain security classification. Or it might be a prepayment that gives a certain viewing time. In both these examples and many others, there would be no need for session state to be stored at the key manager, which greatly aids scalability. However, these are design issues for the application using the secure session and are outside the scope of this paper. All further security messaging proceeds between the receiver and its smartcard, which acts as a proxy of the key manager. If the receiver wishes to dispute delivery of certain parts of the stream, another message is required at the end of the session to present the 'delivery note'. This assumes a model where the delivery note isn't treated as the thing that is bought. Instead it is merely the thing that can potentially prove the items delivered are deficient compared to the items bought.

In section 2, we discuss requirements and describe related work on multicast key management, non-repudiable receipting and detection of re-multicast. In section 3 we describe the underlying composition of the systems we have implemented to meet all our requirements. Then, in section 4, we describe the design of specialisations we have implemented to achieve each requirement. These consist of zero side-effect multicast key management; 'delivery notes' that can even prove timely delivery; and creation of an audit trail using watermarking (yet to be implemented). We also describe how the schemes can be extended for multiple sender scenarios. Section 5 describes how secure sessions are set-up, torn down or modified (e.g. for an unplanned eviction). Section 6 describes various recommended schemes for generating and addressing key sequences and section 7 briefly describes our implementation. Finally limitations of the approach are discussed followed by conclusions.

2. Background and requirements

When using Internet multicast, senders send to a multicast group address while receivers 'join' the multicast group through a message to their local router. For scalability, the designers of IP multicast deliberately ensured that any one router in a multicast tree would hide all downstream join and leave activity from all upstream routers and senders [Deering91]. Any router's own 'join' to the tree is therefore the logical OR of the joins of all downstream receivers. Thus a multicast sender is oblivious to the identities of its receivers. Clearly any security relationship with individual receivers is impossible if they can't be uniquely distinguished. Conversely, if receivers have to be distinguished from each other, the scalability benefits start to be eroded.

2.1 Multicast key management

Because of deliberate receiver anonymity, the potential for eavesdropping Internet multicasts without detection is high. If a multicast sender wishes to restrict its data to a set of receivers, it will typically encrypt the data at the application level. End-to-end access is then controlled by limiting the circulation of the key. A new receiver could have been storing away the encrypted stream before it requested to join the secure session. Therefore, every time a receiver is allowed in, the key needs to be changed (termed backward security [McGrew98]). Similarly, after a receiver is thrown out or requests to leave, it will still be able to decrypt the stream unless the key is changed again (forward security). These problems apply whether the join or leave is initiated by the receiver (e.g. pay TV) or by the host of a secure session (e.g. private side-meetings in a conference). The efficiency of traditional multicast cannot apply if each receiver is being sent different data[i]. Therefore, most approaches work on the basis that when the key needs to be changed, every receiver will have to be given a new key. Continually changing keys clearly has messaging side-effects on other receivers than the one joining or leaving.

The most stringent requirements in the literature involve hundreds of joins and leaves per second for large-scale multicast simulations [RFC2502]. However, this isn't directly relevant, as it refers to the membership volatility of the multicast groups, not necessarily of the 'secure multicast sessions'. We define a 'secure multicast session' as the set of data that a receiver *could* understand, having passed one access control test. If one key is used for many related multicast groups, they all form one secure session. If a particular receiver leaves a multicast group then re-joins but she could have decrypted the information she missed, the whole transmission is still a single secure session. Nonetheless, we are envisaging potentially much larger receiver communities. With a potentially global audience, ten million viewers of a popular Internet pay-TV channel is not an unreasonable requirement. If per-second charging for impulse viewing were possible, it is likely that product managers would find many and various ways to use it in their commercial offerings. Thus, even if just 10% of the audience tuned in or out within a fifteen minute period, this would potentially cause thousands of secure joins or leaves per second.

Incidentally, in the above example, we took one second as the minimum granularity of charging. We use the term 'application data unit' (ADU) as a more general term for the minimum useful atom of data from a security or commercial point of view. ADU size is application dependent. It may be an initialisation frame and its set of associated 'P-frames' in a video sequence or it may be ten minutes of access to a network game (on the grounds that any shorter access would have just as little value). An ADU may be only partially encrypted with the remainder in the clear. This is a common technique used to improve performance while still degrading the information sufficiently to render it useless without decryption [Kunkel97]. ADU size can vary throughout the duration of a stream dependent on the content. ADU size is a primary determinant of system scalability. If a million receivers were to join within fifteen minutes, but the ADU size was also fifteen minutes, this would only require one re-key event.

However, reduction in re-keying requirements isn't the only scalability issue. In the above example, a system that can handle a million requests in fifteen minutes still has to be provided, even if its output is just one re-key request to the senders. With just such scalability problems in mind, many multicast key management architectures introduce a key manager role as a separate concern from the senders. This deals with policy concerns over membership and isolates the senders from much of the messaging traffic needed for access requests.

Ever since Internet multicast became a feasible proposition, scalability improvements have been sought over the earlier group key management schemes that scale linearly with group size (e.g. Ingemarsson *et al* [Ingemar82]). A second class of solutions extend Diffie-Hellman public key cryptography in order to act as a group key, but they require numbers of public key operations that also scale linearly with group size. Ballardie suggested exploiting the same scalability technique used for the underlying multicast tree, by delegating key distribution along the chain of routers in a core based multicast routing tree [IETF_RFC1949]. However, this suffers from a lack of end-to-end security, requiring edge customers to entrust their keys to many intermediate network providers. The Iolus system [Mittra97] sets up a similar distribution hierarchy, but only involving trusted end-systems. However, it also addresses the side-effects of re-keying the whole group by requiring the gateway nodes in the hierarchy to decrypt and re-encrypt the stream with a new key known only to their local sub-group. This introduces a latency burden on every packet in the stream and requires strategically placed intermediate systems to volunteer their processing resource.

An alternative class of approaches involves a single key for the multicast data, but a hierarchy of keys under which to send out a new key over the same multicast channel as the data. These approaches involve a degree of redundant re-keying traffic arriving at every receiver in order for the occasional message to arrive that is decipherable by that receiver. The logical key hierarchy (LKH) [Wallner97] gives each receiver its own key then creates the same number of extra keys, one for each node of a binary tree of keys with each member's key at the leaves. For n receivers, LKH therefore requires the centre to store $2n$ keys but each receiver need only store $\log(n)$. The root of the tree is the group key under which data is encrypted. When a member joins or leaves, all the keys on their branch to the root are replaced in one long message multicast to the whole tree. Each new key is included twice; each encrypted under one of the two keys below it. Each re-key message is therefore $2\log(n)$ times the key length. The closer a receiver is in the logical tree to the changed leaf key, the more decrypt operations it will need to extract its new keys from this message, the maximum being $\log(n)$. Perlman has suggested an improvement to LKH, termed LKH+, where a one way function could be used by all those with knowledge of the existing key to compute the next one [Perlman]. The joining member would only be told the new key and not be able to work back to the old one. Unfortunately, the same argument could not be applied for backward security.

The one-way function tree (OFT) technique is in the same class of approaches [McGrew98]. Incidentally, OFT is also presented in [Balen99], which is particularly notable for its comprehensive and accurate review of the literature. Like LKH, all members have their own key, and a binary tree of keys is built over them with the root also being the group key. However, the keys at each intermediate node are a combination of the hashes of the two keys below, rather than being freely generated. Thus Perlman's suggestion cannot be applied to OFT because the group key is not independent of the keys lower in the tree. As a result, LKH+ becomes more efficient than OFT in most scenarios. The standardised approach to pay-TV key management also falls into this class [ITU-R.810]. A set of secondary keys is created and each receiver holds a sub-set of these in tamper-resistant storage. The group key is also unknown outside the tamper-resistant part of the receiver. In case the group key becomes compromised, a new one is regularly generated and broadcast multiple times under different secondary keys to ensure the appropriate receivers can re-key.

All work in this class of approaches uses multicast itself as the transport to send new keys. As 'reliable multicast' is still to some extent a contradiction in terms, all such approaches have to allow for some receivers missing the occasional multicast of a new key due to localised transmission losses. Some approaches include redundancy in the re-keying to allow for losses, but this reduces their efficiency and increases their complexity. Others simply ignore the possibility of losses, delegating the problem to a choice of a sufficiently reliable multicast scheme.

Dillon [Dillon97] falls into a separate class of approaches to key management. Documents transmitted over satellite are encrypted with a key generated from a one way hash of a seed keyed with the document ID. The seed is stored at the receiver in a tamper-resistant security engine having been transmitted under the public key of that security engine, the private key being installed in the security engine at manufacture. To request a document from a catalogue, the receiving computer requests that a key be generated by its associated security engine which it pre-loads into its satellite receiver, to be used when the next encrypted document broadcast is scheduled. Any documents that arrive with no corresponding key awaiting them are discarded. The distinguishing feature of this class of solutions is that the group key used to encrypt a document is specific to that document ID. Thus each broadcast document is encrypted using a different key rather than the key only being changed in synchrony with the addition or removal of receiver interest.

In the interests of full disclosure, we note that the present work, which falls into the same class as Dillon, is described in a European patent filing [Briscoe97].

2.2 Multicast non-repudiation

The need for proof of delivery is recognised in two taxonomies of multicast security requirements [Bagnall99, Canetti98], but solutions are rarely discussed in the academic literature. Proof of delivery is a very different problem to acknowledgement of delivery. It has to be possible to prove the receiver did indeed receive data when they might deny reception. Pay-TV and pay-per-view systems invariably use the tamper-resistant processing and storage capabilities of the local receiver to record which products or programmes have been requested in order to form a bill at a later time (e.g. [ITU-R.810, Dillon97] as already cited).

The novel aspect of the present work is the ability to prove that individual fragments of an isochronous stream (e.g. video) not only arrived, but arrived in time to be played out, giving suitable perceived quality for a real-time application. Our approach is for the receiving system to only request a key to decrypt the stream if there is sufficient time remaining to decrypt it and still achieve smooth play-out. Because the tamper-resistant key generator records how many keys have been requested, it can be used to resolve disputes over proof of timely delivery. This is possible because the link between the receiving computer and the smartcard has predictable latency and minimal risk of packet drop compared to the Internet connection between sender and receiver. Thus, when the smartcard responds to a key request this can be assumed to equate to a successful use of the key with a high degree of certainty.

2.3 Multicast audit trail

Re-multicast of received data requires very low resources on the part of any receiver. Currently sending to a multicast is offered by just a few Internet providers at a considerable price premium. However, once the market becomes competitive we can assume that prices for sending and receiving multicast will reflect costs plus a margin. Therefore sending prices will be nearly as low as for receiving. Thus, even if the value of the information received is relatively low there is always a profit to be made by re-multicasting data and undercutting the original price, as proved in Herzog *et al* [Herzog95]. The digital signet approach [Dwork97] is appealing, in that copy-prevention is self-enforced. However, digital signets are intended for scenarios where the data volume is too large to be efficiently transferred over a network and has instead been previously distributed by other means (e.g. CDs or DVDs). The approach relies on the key size being as large as the data, unless the copier divulges a personal secret. In the scenarios we are considering, the data is already being transferred over a network, therefore digital signets would give no protection.

In general, prevention of information copying is considered infeasible; instead most attention focuses on the more tractable problem of copy detection. Early work in the field of digital steganography showed that it was possible to 'watermark' different copies of a copyrighted digital work. If a watermarked copy was later discovered, it could be traced back to its source, thus deterring the holders of original copies from passing on further, illicit copies [Brin94]. Watermarks are typically applied to the least significant bits of a medium to avoid significantly degrading the quality. Such bits are in different locations with different regularity in different media, therefore there is never likely to be a generic approach [Schneier96]. The most generic scheme discussed to date is Chameleon [Anders97]. In Chameleon a stream is ciphered by combining a regular stream cipher with a large block of bits. Each receiver is given a long-term copy of the block to decipher the stream. In the concrete example given, four 64b words in the 512kB block are chosen by indexing the block with the output of the regular stream cipher. Then all four are XOR'd together with each 64b word of the stream. The block given to each receiver is watermarked in a way specific to the medium. For instance, the least significant bit of every 16b word of an audio stream might be the only place where a watermark can be stored without degrading the content significantly. Because the block is only used for the XOR operation, the position of any watermarked bits is preserved in the output.

Naor *et al* [Naor98] formalises a pragmatic approach to 'traitor tracing' by proposing a parameter that represents the minimum number of group members that need to collude to eliminate a watermark. The elimination criteria are that none of the conspirators are identifiable, and it is assumed that the copyright owner will want to avoid accusing innocent members. For instance, changing at least the square root of the total number of bits that could hold a watermark in the Chameleon scheme would protect against conspiracies of four or less members. Anderson and Manifavas point out that it is probably easier for an attacker to subscribe in a false name than organise larger conspiracies, therefore this seemingly small protection against collusion should at least raise the strength of the weakest link to that of the second weakest.

Watercasting [Brown99] is a novel, if rather convoluted way to embed an individual watermark in each receiver's copy of multicast data. Multicast forwarding is modified by including active networking elements at strategic branch points. These elements drop redundant data inserted into the original stream in order to produce a different drop pattern on each forwarded branch. A chain of trusted network providers is required for watercasting, each of which has to be willing to reveal their authenticated tree topology to each sender.

In this paper, for completeness, we report how it is possible to add an audit trail back to the copier of multicast information using watermarking. Our approach is not novel in this respect, simply re-using Chameleon. However, we include it to demonstrate our modular approach to the addition of mechanisms.

2.4 Other requirements

Although the concerns that led to this work were primarily commercial, the techniques apply equally in scenarios where the information is considered sensitive rather than for sale. Both individual control of access and identification of responsibility for information leakage are important requirements in such scenarios. However, proof of receipt is rarely required. Fortunately, the proposed solutions are deliberately modular. Therefore if receipting isn't important, that module need not be used.

Beyond the three requirements we have focussed on so far, the taxonomies we have already cited include many other possible combinations of security requirements for multicast. We have placed sender authentication outside the scope of this paper, but its importance merits a brief survey of the literature. A sender may merely need to prove it is one of the group of valid receivers in which case use of the group encryption key suffices. If the group valid senders is different from that of the receivers, a message authentication code (MAC) on all sent data based on a single key shared by the senders is sufficient and efficient. If receivers require each sender to authenticate their messages individually, public key signing leads to an unscalable solution because of the sheer volume of heavy asymmetric key operations required. Balenson *et al* [Balen99] and Canetti *et al* [Canetti98] both provide up to date reviews of more efficient approaches to this problem.

So far we have focussed on the scenario where the data is an ordered stream and access is given between some start and some later end point. A more random access approach might be required for non-sequential application name spaces [Fuchs98], for which we present a brief solution at the end of our section on key generation techniques.

2.5 Implementations

Many of the schemes discussed above are theoretical works. Known exceptions are Iolus and Chameleon. A report on implementation experience with LKH is provided by Boxall [Boxall98] and Shoup *et al* report on their implementation of session key distribution using smart cards [Shoup96].

3. Basic Scheme

In explaining the basic scheme we will firstly give a concrete example of how it would be used and then give a description of the core of the scheme upon which other features can be built.

3.1 A Concrete Example

Our example is of a content provider who wishes to multicast streamed video and charge viewers for watching it.

The first thing the content provider needs to do is to divide the video stream into units that potential viewers can use to select what they want to see. The obvious unit here is something like the "TV programme", i.e. a complete film or news broadcast or an episode in a series. Each of these units is given an ID. Next the content-provider sets up a video server which has access to the video in a streamable form. As part of the set-up process a seed is generated and a formula chosen. This is used to generate symmetric encryption keys based on this seed for encrypting and decrypting the data. The content provider also sets up another key management server to hand out these seeds in return for payment. The sender passes the programme information, including the seed and formula, on to that server. The content provider then advertises the programme, perhaps using a web site or email, with the programme ID and the key management server being used to uniquely identify the programme to the system. When the time comes to broadcast the video the video server starts streaming. Each frame of video is given it’s own ID within the programme and a corresponding key is generated from this ID, the seed key and the formula. This new key is used to encrypt each frame before it is sent.

Now we consider the receiver’s side. The user has a computer that is connected to the network and a smartca reader. They also have a smartcard which contains it’s own public/private key pair and has been certified by a trusted third party. The private key is unavailable to the user. The user finds a programme they want to watch on a web site and clicks on the "set-up" link for that programme. The link URL downloads a file containing the information that is needed to join the session and the browser passes this on to the user’s video player software (which has been configured as a browser helper application). The video player passes this information on to a socket factory, the internals of which are outside the scope of this paper - see Flexinet [Hayton98]. The essential point is that a communications stack is built containing a decrypter. When the decrypter is set up it in turn sets up a key generator in the smart card, which in turn needs a seed and a policy. The decrypter requests these from a key server in return for a payment. They arrive encrypted with the smartcard’s public key and are passed to the key generator. All the references necessary for this are included in the original downloaded file.

The smartcard now has all the information it needs to generate the keys to decrypt the frames. The socket factory then passes a socket reference back to the video application which need not be aware that decryption is taking place beneath it. The video application simply uses this socket to join the multicast. When the TV programme starts, the socket waits until it receives all the data for each frame, then asks the smartcard for the key for that particular frame, decrypts the frame and passes the frame on to the video player application for decompression and display. The smartcard can record the number of keys that were generated per programme and, space permitting, which keys were passed out. If any frames are received for other sessions the socket knows not to ask the smartcard for keys, and even if it did the smartcard would not provide them.

After the programme has finished, there is no need to do anything further unless, for some reason, the receiver did not receive all the video. What the receiver can do is to ask the smartcard to produce a ’delivery note’ for the partially received programme which the smartcard signs with it’s private key. This can be forwarded on to the payment server to prove that not all the data was received and a refund can be arranged.

3.2 Core Set-up

The core scheme involves a sender sending data via some distribution mechanism to zero or more receivers. The first thing to note is that during the send phase there is no need for feedback from the receiver and the number of receivers is irrelevant to the functioning of the sending stage. The sender divides the data stream into a number of application data units (ADUs). As we have discussed previously, we take each of these ADUs to be a minimum useful component of the data stream, such as a single video frame or frame element for video. Each ADU sent in a session has an ID associated with it. These IDs will probably be numeric but there is no requirement that they should be or that they follow a particular sequence. For the session there exists a mapping of IDs to keys and, before it is sent, the data in an ADU is encrypted using the key associated with the ADU’s ID. Any receiver receiving data in the session must know the ID to key mapping used for that session and uses it to find the key for any ADUs it receives and wishes to decrypt.

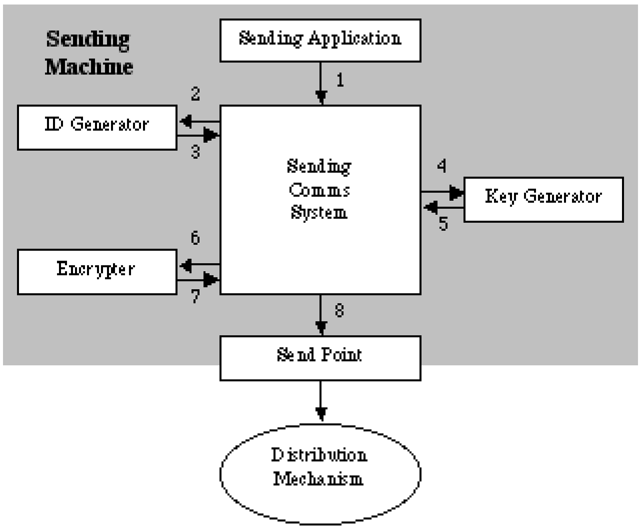

Figure 1 - Sending Stage - Sender

The process of sending data is as follows:

1. Sending application passes an unencrypted ADU on to the communications system.
2. The communications system requests the next ID from the ID generator...
3. ...which returns a new ID.
4. The communications systems requests the key for that ID from the key generator...
5. ...which returns the key.
6. The communication system passes the ADU and the key on to the encrypter...
7. ...which returns an encrypted ADU.
8. The communications system passes the encrypted ADU and the ID on to the send point for distribution.

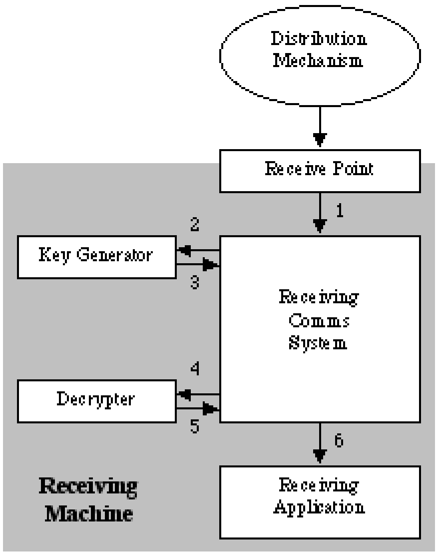

Figure 2 - Sending Stage - Receiver

The process for receiving data is as follows (note that smartcard security is only added when we discuss the variations later):

1. The receive point passes on an encrypted ADU and ID it has received from the distribution mechanism to the communications system.
2. The communications system requests the key for that ID from the key generator...
3. ...which returns the key.
4. The communications system passes the encrypted ADU and the key on to the decrypter...
5. ... which returns an unencrypted ADU…
6. …which is passed on to the ADU to the application for processing.

4. Variations

4.1 Multicast Key Management

The scenario here is the situation where we have an ongoing multicast session and where receivers joining the session are only allowed to receive a portion of the data. An example of this might be where the multicast was a video broadcast and where a receiver might pay to receive an hour or a days worth of video.

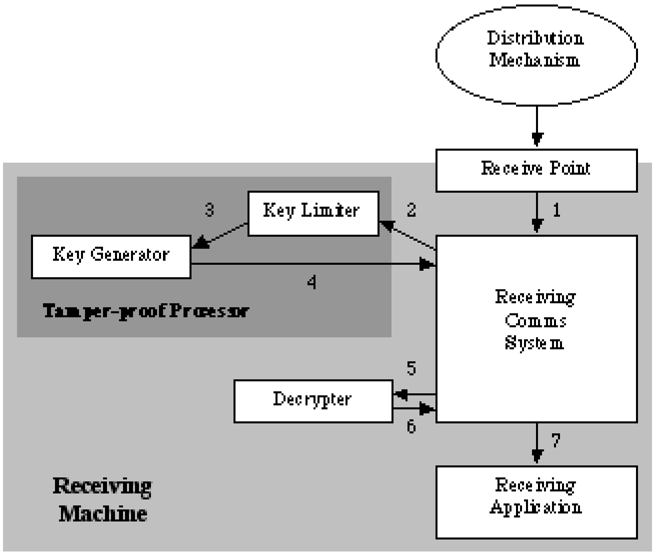

Figure 3 - Multicast Key Management

In this case we add a key limiter that limits the production of keys, i.e. to a certain number or perhaps to a particular range of IDs. The key limiter and the key generator are placed within the tamper-proof processor.

The steps are as follows:

1. The receive point passes an encrypted ADU and ADU ID on to the receiving communications system.
2. The communications system passes the ID on to a key limiter.
3. *If* the ID passes the key limiter’s test, it is passed on to the key generator, *otherwise* the process stops here.
4. The key generator passes the key for the ID back to the communications system.
5. The communications system passes the encrypted ADU and the key to a decrypter…
6. …which returns the unencrypted ADU…
7. …which is passed on to the application.

The limiter will usually also be required to restrict its output to one response per key. This protects against the same card being shared around multiple receivers as a relatively convenient way to decrypt the same data multiple times rather than passing all the keys around. This would require internal receipting capabilities similar to those described in the next section.

4.2 Non-repudiation

In this scenario we are concerned with being able to confirm how much data an application received. Sending acknowledgements for each ADU is impractical, especially as the number of receivers grow large. Also, this does not prevent the receiver trying to fool the sender by not sending acknowledgements for ADUs it has received. What we do in this case is to produce a 'delivery note' of all the data received in a session. If, at the end of a session, we need to confirm how much data was received by an application in a particular session, we can query its secure processing environment and get the 'delivery note' for that session.

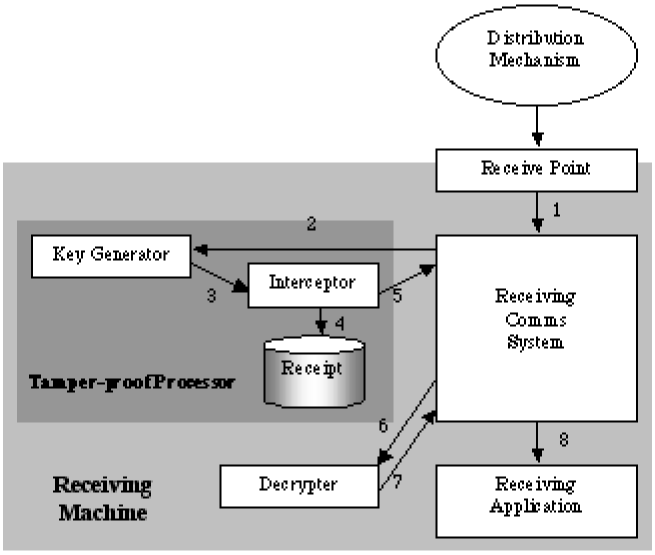

Figure 4 - Non-repudiation

In this case we can create receipts for every ADU decrypted by intercepting every return of a key and recording it in a file. The steps are as follows:

1. Receive point receives an encrypted ADU and ADU ID from distribution mechanism and passes it on to the communications system.
2. The communications system passes the ID on to the key generator…
3. ...which passes the key and ID on to the interceptor…
4. …which records the ID on a receipt…
5. …and passes the key back to the communications system.
6. The communications system passes the encrypted ADU and the key on to the decrypter…
7. …which returns the unencrypted ADU to the communications system…
8. …which passes it on to the application.

For ordered streams, the receipting storage format only needs to be a simple index to the last key given out. If random access to any ADU is envisaged, a block of bits, one for each ADU, would be required to record which keys had already been given out. More efficient tree-based variants are possible to reduce storage requirements in most realistic scenarios.

If different types of ADUs in a stream require different treatment with respect to security it is simplest to create a separate secure session for them. For instance, high quality transmission costs for adverts might be refunded only if a delivery note is returned to prove they were at least decrypted if not watched (e.g. a hash of the decrypted frames might be required). These would form a sub-session with a different policy in the smartcard.

4.3 Audit Trails

The problem this variation helps to address is that of a receiver in the session colluding with other receivers that are not part of the session by sending them keys or decrypted data. There are two variants: on-card and off-card watermarking, depicted in Figures 5a and 5b respectively. In the first variant only the plaintext data is watermarked therefore each ADU key is never revealed outside the smart card. In the second variant, the keys themselves are watermarked so both the keys and the data can be revealed outside the card.
If the watermarked keys or data are then sent on to other machines and detected later then it would be possible to get the identity of the source of the "leak" from the watermark. This variation assumes that the data is watermarkable, e.g. images.

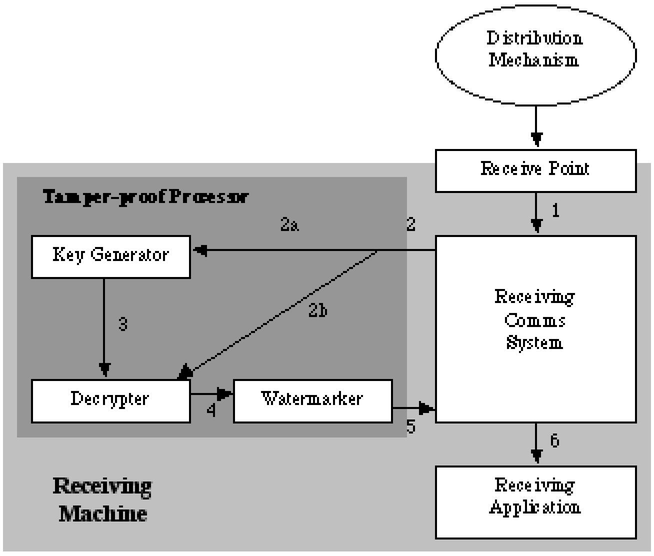

Figure 5a On-card Watermarking

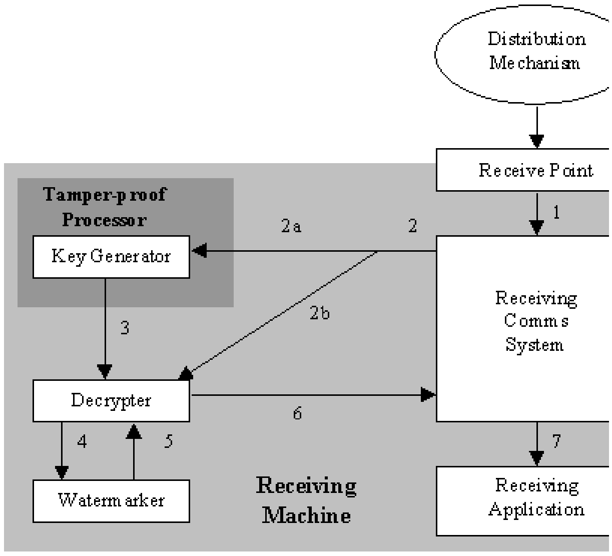

Figure 5b Off-card Watermarking

The steps for on-card watermarking are as follows:

1. The receive point passes the encrypted ADU and ADU ID into the communications system.
2. The communications system passes a) the ID into the key generator and b) the encrypted ADU into the decrypter.
3. The generator passes the key for that ID into the decrypter.
4. The decrypter uses the key to decrypt the ADU and passes it to the watermarker.
5. The watermarker adds a watermark to the ADU and passes the watermarked ADU on to the communications system…
6. …which passes the watermarked ADU on to the application.

On-card watermarking is only feasible with a fairly highly powered tamper-resistant cryptographic co-processor. It is impractical with smartcards due to processing and memory limitations. Off-card watermarking needs only light card resources. An approach such as Chameleon [Anders97] as described earlier is preferred as long as there is sufficient memory on the receiver to hold the whole watermarked key block (about 512kB in the concrete example). The following steps for off-card watermarking assume the sender encrypter unit produces its stream cipher by combining a standard cipher with an unwatermarked version of the long-term key-block, as in Chameleon.

The steps at the receiver for off-card watermarking are as follows:

1. The receive point passes the encrypted ADU and ADU ID into the communications system.
2. The communications system passes a) the ID into the key generator and b) the encrypted ADU into the decrypter.
3. The generator passes the intermediate key for that ID into the decrypter.
4. The decrypter passes the intermediate key to the watermarker
5. The watermarker uses the intermediate key as an index into a long term watermarked key block to return the key to the decrypter
6. The decrypter uses the key to decrypt the ADU and passes it to the communications system…
7. …which passes the watermarked ADU on to the application.

4.4 Multiple Sender Systems

This variation addresses the issue of having many senders within a session. For simplicity's sake we might want to use the same key generator for information sent from all senders, although this would require that the key generator would be able to generate keys for any order of IDs (which would be true in the general case of senders not being synchronised). If we wished to have key generators that required IDs in order or we wished to produce individual delivery notes for each sender (see Non-repudiation) then we need to maintain a number of key generators, one for each sender. To identify each sender we would have to generate a unique ID for each one, i.e. for information sent across the Internet we could use the IP address and port number which is sent as part of the packet. To seed the sequences we can then use a common seed for all senders within a session which is then combined with the unique ID in some way, i.e. XORed with the common seed, which is then used as the seed for that sender. The receiving stack now uses a switch to retrieve the correct key for the data unit. Of course, the sending stack need only maintain a single key generator for all data it sends to a session.

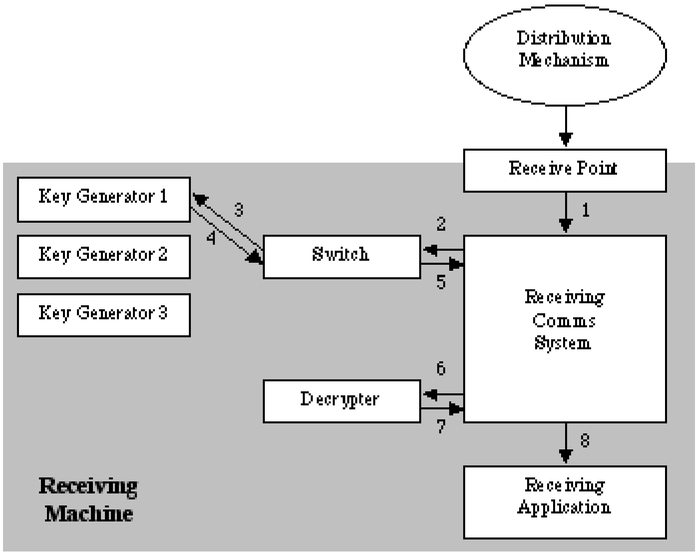

Figure 6 - Multiple Sender Systems - Receiver

The steps involved are:

1. The receive point passes an encrypted ADU, ADU ID and source ID on to the communications system.
2. The communications system passes the ADU ID and source ID on to the switch…
3. …which uses the source ID to pass the ADU ID on to the correct key generator…
4. …which returns the key for that ID…
5. …which the switch returns to the communications system.
6. The encrypted ADU and key are passes to the decrypter…
7. …which returns the unencrypted ADU…
8. …which is then passed to the application.

5. Session Control

For any of the above schemes can be used it is necessary to have some auxiliary functions implemented.

In the follow sections this notation is used:

1. *sign(k,d)* - *d* signed with key *k* (i.e. *d* and the signature of *d* with *k*)
2. $enc_a(k,d)$ - *d* encrypted asymmetrically with key *k*
3. $enc_s(k,d)$ - *d* encrypted symmetrically with key *k*

5.1 Tamper-proof Processor Confirmation

The object here is to confirm that the tamper-proof processor is one that the sender can trust. We assume that every secure processing environment leaves the factory with a securely stored private key and a public key that has been signed by a trusted third party (TTP) trusted by the sender.

1. Sender generates a random string *r* (a nonce)
2. Sender sends *r* to receiver
3. Receiver sends *r* to secure space
4. Secure space signs *r* with private key *s* to produce *sign(s,r)*
5. Secure space returns *sign(s,r)* and public key *p* signed with the TTP's private key *t* (producing *sign(t,p)*) to receiver
6. Receiver returns *[sign(s,r), sign(t,p)]* to sender

7. Sender checks TTP is one it trusts
8. Sender checks *sign(t,p)* with TTP (either by invoking TTP server or using cached TTP public key)
9. Sender checks *sign(s,r)* with *p*.

5.2 Session Set-up

The sender needs to set-up the keying system so that it can generate a sequence of numbers for decoding each packet. This sequence will be some chaotic/pseudo-random sequence.

1. Sender generates a seed value *v*.
2. Sender generates a session key *k*.
3. Sender encrypts *v* using secure space's public key *p* producing $enc_a(p,v)$.
4. Sender sends $[k,enc_a(p,v)]$ to receiver.
5. Receiver sends $[k,enc_a(p,v)]$ to secure space.
6. Keying system sets packet counter to zero.
7. Keying system deciphers $enc_a(p,v)$ using secret key *s*.
8. Keying system initialises sequence generator with *v*.

For multicast key management the sender will also send some information to limit the production of keys, such as a limit on the maximum number of keys.

This describes a simple scenario where a single sequence generator can create an unlimited sequence of numbers and create a single delivery note type. More realistically the session information would include:

Sent in plain:

- Session Key

Sent encrypted:

- Seed value
- Sequence generator type
- Delivery note type (for non-repudiation)
- Maximum number of keys to generate (for multicast key management)

In this scenario there are a limited number of sequence generators and delivery note types that can be used as it is identifiers that are being sent over as part of the session information. Alternatively a secure class loader could be implemented that would allow new sequence generators and delivery note types to be uploaded into the encryption system. This would offer the most future-proofing.

Another aspect of session set-up is session amendment. The user may pay to receive a certain amount of data and then later on pay for some more. This would ideally be handled by updating the session information (probably just increasing the maximum number of keys to be generated) while the session is active.

5.3 Session Tear-down

Ending sessions isn't a big issue with this system: sessions will end when the sender stops sending data or the key generator stops generating keys. In the case of non-repudiation though there is a need to retrieve the delivery note from the secure environment. The following steps allow this.

1. Receiver requests delivery note for session key *k* from keying system.
2. Keying system generates delivery note for session key *k*, c_k.
3. Keying system signs c_k with private key *s* giving $sign(s,c_k)$.
4. Keying system returns $sign(s, c_k)$ to receiver.
5. Receiver sends $sign(s, c_k)$ to sender.
6. Sender checks $sign(s, c_k)$ against public key *p* of keying system known to be used by the receiver (database

lookup).
7. Sender refunds if necessary.

5.4 Access Revocation

It may be desirable for a session controller to be able to modify or revoke a receiver's membership of a session. Detailed below are two methods of adding such control. Both solutions assume that each member of the session has an ID (or several IDs) within the session, although this ID does not have to be unique to the member (if it is not unique then the ID obviously represents a group). The solutions also assume that control data is passed into the secure processing environment.

5.4.1 Separate Control Stream

This approach is the more heavyweight of the two methods but allows the sender to guarantee that receivers can receive no more data. When an ADU arrives the secure space is sent an encrypted block as well as a key ID. It decrypts the data. If the secure space ID(s) contained in the encrypted block refer to this particular space then it checks the flags. If the stop flag is set then no more keys are passed out. If the contact sender flag is set then the secure space does a remote procedure call to the sender (or the sender's representative) and gets a new key generation policy.

ADU format:

1. Signature of Hash (2)
2. Hash of 3, 4, 5, 6
3. ADU ID
4. Stop flag (y/n) (encrypted)
5. Contact sender flag (y/n) (encrypted)
6. secure space IDs (encrypted)
7. ADU data (encrypted)

The stack passes 1, 2, 3, 4, 5 and 6 into the secure space to receive the key for 7.

If the length of the control message and number of secure space IDs is variable then there needs to be an unencrypted field before the control message stating the total length of the control message and secure space IDs.

5.4.2 Poison Pill

This is a probabilistic approach. Every time an ADU is sent it contains an encrypted control message and secure space ID which must be passed into the secure space along with the key ID to obtain the key. The control message may be a code representing the command "commit suicide". If the secure space receives this and the secure space ID(s) relate to it, it executes the command.

If several users need to be thrown out of the session then their secure space IDs will be rotated through different packets.

ADU format:

1. Signature of hash (2) (signed with sender's public key)
2. Hash of 3, 4, and 5
3. ADU ID
4. Control message (encrypted)
5. secure space ID(s) (encrypted)
6. ADU data (encrypted)

The stack passes 1, 2, 3, 4 and 5 into the secure space to receive the key for 6.

If the length of the control message and number of secure space IDs is variable then there needs to be an unencrypted field before the control message stating the total length of the control message and secure space IDs.

6. Key Generation

We consider the key generation algorithms to be of two classes: ordered key generation and out-of-order key generation.

6.1 Ordered Key Generation

Ordered key generation involves using some sequence that can only generate numbers in a particular order, i.e. we cannot generate a number at position *n* in the sequence if we have generated the number at position *m* where $m > n$, unless we store all numbers in the sequence up to the *m*th term or regenerate the sequence. We consider storing numbers in the sequence or regenerating the sequence impractical for devices that are as limited as smartcards.

Examples of ordered sequences are chaotic sequences.

6.2 Out-of-Order Key Generation

Out-of-order key generation involves having some mapping that can map any valid ID to a number. The order that the IDs arrive now becomes irrelevant.

An example of out-of-order key generation is as follows. Use the seed to generate three numbers, all the same number of bits as the number required. One of these numbers in a new seed, *s*, and the other two we'll refer to as *a* and *b*. To generate a new number from an ID we go through each bit in the ID in turn. If the bit is a 0 we XOR *s* with *a* and left-shift the bits of the result, otherwise it is XORed with *b* and right-shifted. The resulting number is returned.

7. Implementation

An implementation of this system was created for demonstration purposes. It was written in Java 1.1 and used the Cryptix 3.0.3 [Cryptix] library for cryptography. Aspects of the system can be easily changed: the formula used to generate keys (one based on the logistic mapping [Gleick87] has been implemented); the policy for limiting keys (policies for producing fixed numbers of keys and keys for a range of IDs have been implemented); the cryptographic system (DES was implemented); the receipt type (one simple receipt was implemented). The system did not use any smartcards but those aspects of the design are cleanly separated from the rest of the system. Simple graphical applications were written to demonstrate the sender and receiver roles.

8. Limitations and further work

Our approach relies on the tamper-resistance of smartcards. Products are continually being produced with improved tamper-resistance features, but there will always be attrition between the designers of tampering techniques [Anders96] and the designers of resistance to them. The need to regularly replace the smartcard is therefore an inherent weakness in our scheme. Indeed, the fact that a smartcard is needed at all, is in itself a major impediment to take up of the scheme. We have tried to mitigate this barrier by designing for a generic trusted third party card (e.g. a Java card), rather than one tailored to a specific service provider.

The non-repudiation aspect of this work is only useful in a commercial model where there is an incentive for the receiver to volunteer the delivery note to the information provider. Such scenarios are easy to create (a couple are given as examples in the introduction), but this means the capability is not universally useful. For instance, it would not be possible to give away the stream of information then ask each receiver to volunteer their delivery note to calculate how much they should pay. The good corollary of this is that it is difficult to get the smartcard to give out thousands of keys off-line in order to break the seed. The smart card won't give out any keys if it doesn't have a key limiter policy and if it does have a policy, it will only give out keys the user has paid for.

This paper contains no formal security analysis of the strength of the schemes employed. A number of questions are left unanswered, such as whether the seed of a pseudo-random sequence becomes easier to predict, the more values from the sequence are revealed.

We must also admit to the standard limitations that apply to most other work on copyright protection. A watermark-based audit trail is only proof against small numbers in collusion and it only helps detection not prevention. Also, traitor tracing relies on finding the watermarked data in the first place; a problem that this paper and others on the subject invariably leave unresolved.

Regarding further work, we claimed in the abstract that this approach could be applied to other means of bulk data distribution than multicast, such as DVD (digital video/versatile disk). We envisage a scenario where data on the DVD would be encrypted with a stream cipher such that it would be indistinguishable from a multicast stream once it was read from the disk. As long as the initial set up with the smartcard had occurred on-line, the rest of the DVD could be played off-line, only requiring interaction with the smartcard, not the network. Any final 'delivery note' of exactly what had been accessed would then be available to present to the provider of the DVD. In a similar vein, policies and seeds to load into the smart card could be supplied on various media other than over the Internet. All these scenarios and more are introduced in [Briscoe97], but we have done no specific design or implementation work on them.

9. Conclusion

We have presented a number of modular mechanisms to enable secure sessions tailored to each individual multicast receiver while at the same time not compromising the inherent scalability of Internet multicast, achieved through loose coupling between senders and receivers. Unlike other schemes, we typically require absolutely no coupling at all from receivers back to senders but still create a security relationship between each receiver and a key manager replica. The key managers can be highly replicated as they require no coupling back to the sender. As long as a stateless commercial model is required (e.g. pre-payment rather than credit), key manager replication is limitless. Further, as members join and leave, there are absolutely no side-effects on other receivers, unlike traditional multicast key management schemes.

All this loose coupling is made possible by a simple technique where multicast senders systematically change the group encryption key rather than only changing it whenever there is a change to the group membership. This innovation is driven by the insight that there will always be a minimum application data unit (ADU) granularity, within which there is no commercial advantage to changing the group key. The traditional approach has been to group together membership change events within the timespan of an ADU and then drive key changes dependent on whether none or some events have occurred within each timeslot. Instead, by systematically changing group keys whether or not it is necessary, the whole system can rely on the key changes and not require tight coupling back to the senders. A further advantage of this approach is that there is no need to send any control messages over the multicast channel itself. Thus no reliable multicast mechanism is assumed or required and no complexity is involved when messages are lost. The only exception is the rare need to send a 'poison pill', which merely requires statistical delivery.

In order to distribute the load of key management further, we require each receiver to operate a smartcard, into which the information provider can install a key generator capable of mirroring the systematic key generation of the senders. We prefer generic smartcards certified by trusted third parties, so that any key generator can be installed at session set-up. This mitigates the barrier created by the need for each receiver to obtain a card, as it can be re-used for multiple services. A policy is installed into the smart card at session set up to control which keys it will give out to its receiver. The details depend on the specifics of the wider application.

Further, we have shown by implementation that it is even possible to prove timely reception of real-time data units using this arrangement. The smart card records which keys it has been asked for and if a packet arrives late, the receiver simply refrains from asking for the key. Thus, the smartcard generates a delivery note that can later be used by the receiver to prove that only a certain number of data units were usefully decrypted.

We have also described how it would be possible to combine the above approach with a key watermarking scheme such as 'Chameleon'. This provides a small but significant deterrent against a receiver giving away or re-selling either the keys or the decrypted data, because both are watermarked in such a way as to trace that receiver.

We believe these mechanisms (combined with sender authentication approaches described elsewhere) provide a soundly engineered basis for a number of very large scale commercial applications built over Internet multicast. We have also briefly described how the same techniques could be usefully applied to other bulk data distribution mechanisms, such as DVDs. Applications of the techniques have also been discussed where protection of information security rather than value is required.

10. References

[Anders96] Ross Anderson and MG Kuhn, ""Tamper Resistance - A Cautionary Note", in proceedings of the second USENIX Electronic Commerce Workshop (Nov 96) pp 1-21 <URL:http://www.cl.cam.ac.uk/users/rja14/#Reliability>

[Anders97] Ross Anderson & Charalampos Manifavas (Cambridge Uni), "Chameleon - A New Kind of Stream Cipher" Encryption in Haifa (Jan 1997), <URL:http://www.cl.cam.ac.uk/ftp/users/rja14/chameleon.ps.gz>

[Bagnall99] Pete Bagnall, Bob Briscoe & Alan Poppitt, (BT), "Taxonomy of Communication Requirements for Large-scale Multicast Applications", Internet Draft (work in progress), Internet Engineering Task Force (17 May 1999) <draft-ietf-lsma-requirements-03.txt>

[Balen99] D. Balenson, D. McGrew & A. Sherman (TIS Labs at Network Associates), "Key Management for Large Dynamic Groups: One-Way Function Trees and Amortized Initialization", February 26, 1999, <draft-balenson-groupkeymgmt-oft-00.txt>

[Ballard95] Tony Ballardie and Jon Crowcroft (UCL), "Multicast-specific threats and counter-measures," Proceedings of the Internet Society 1995 Symposium on Network and Distributed System Security 16-17 Feb 1995, San Diego, California, IEEE Computer Society (1995), 2-14. <URL:ftp://cs.ucl.ac.uk/darpa/IDMR/mcast-sec-isoc.ps.Z>

[Boxall98] Sarah Boxall (Cambridge Uni), Report on industrial placement (Sep 1998).

[Brin94] Brin S, Davis J and Garcia-Molina H, "Copy detection mechanisms for digital documents", Department of Computer Science, Stanford University (1994).

[Briscoe97] Bob Briscoe and Ian Fairman (BT), "Multicast Key Management", European patent publication no. EP98304429.8 (Mar 97).

[Brown99] Ian Brown, Colin Perkins & Jon Crowcroft (UCL), "Watercasting: Distributed Watermarking of Multicast Media", submitted to Globecom ’99, Rio de Janeiro, (December 1999), <URL:ftp://cs.ucl.ac.uk/darpa/watercast.ps.gz>

[Canetti98] Canetti, R. & B. Pinkas, "A taxonomy of multicast security issues," Internet Draft (work in progress), Internet Engineering Task Force (May 1998), <URL:draft-canetti-secure-multicast-taxonomy-00.txt>

[Cryptix] Cryptix library, <URL:http://www.cryptix.org/>

[Deering91] S. Deering, “Multicast Routing in a Datagram Network,” PhD thesis, Dept. of Computer Science, Stanford University,
(1991).

[Dillon97] Douglas M Dillon (Hughes), "Deferred Billing, Broadcast, Electronic Document Distribution System and Method", International patent publication no. WO 97/26611 (24 July 1997).

[Dwork97] Cynthia Dwork & Jeffrey Lotspiech (IBM), Moni Naor (Weizmann Inst of Sci, Rehovot), "Digital Signets: Self Enforcing Protection of Digital Information". in Proc. 28th Ann. ACM Symp. on Theory of Computing (1997), <URL:http://www.wisdom.weizmann.ac.il/~naor/PAPERS/signet.ps>

M. Fuchs, C. Diot, T. Turletti, M. Hoffman, "A Naming Approach for ALF Design", in proceedings of HIPPARCH workshop, London, (June 1998) <URL:ftp://ftp.sprintlabs.com/diot/naming-hipparch.ps.gz>

[Herzog95] Shai Herzog (IBM), Scott Shenker (Xerox PARC), Deborah Estrin (USC/ISI), "Sharing the cost of Multicast Trees: An Axiomatic Analysis", in Proceedings of ACM/SIGCOMM '95, Cambridge, MA, Aug. 1995, <URL:http://www.research.ibm.com/people/h/herzog/sigton.html>

[Gleick87] James Gleick, "Chaos", 1987, ISBN: 0749386061

[IETF_RFC1321] Ronald L. Rivest, "The MD5 Message-Digest Algorithm", Request for Comments (RFC) 1321, Internet Engineering Task Force (1992) <URL:rfc1321.txt>

[IETF_RFC1949] Tony Ballardie, "Scalable multicast key distribution", Request for Comments (RFC) 1949, Internet Engineering Task Force (May 1996) <URL:rfc1949.txt>

[IETF_RFC2109] D. Kristol (Bell Labs), L. Montulli (Netscape), "HTTP State Management Mechanism", Request for Comments (RFC) 2109, Internet Engineering Task Force (Feb 1997), <URL:rfc2109.txt>

[IETF_RFC2502] M. Pullen (George Mason Uni), M. Myjak (The Virtual Workshop), C. Bouwens (SAIC), "Limitations of Internet Protocol Suite for Distributed Simulation the Large Multicast Environment", Request for Comments (RFC) 2502, Internet Engineering Task Force (Feb 1999), <URL:rfc2502.txt>

[Ingemar82] I. Ingemarsson, D. T. Tang, and C. K. Wong, "A Conference Key Distribution System", IEEE Transactions on Information Theory, vol. IT-28, pp. 714-720, 1982.

[ITU-R.810] ITU-R Rec. 810, "Conditional-Access Broadcasting Systems", (1992) <URL:http://www.itu.int/itudocs/itu-r/rec/bt/810.pdf>

[Kunkel97] Thomas Kunkelmann, Rolf Reinema & Ralf Steinmetz (Darmstadt Tech Uni), "Evaluation of Different Video Encryption Methods for a Secure Multimedia Conferencing Gateway", 4th COST 237 Workshop, Lisboa, Portugal, Springer Verlag LNCS 1356, ISBN 3-540-63935-7 (Dec 1997), <URL:http://www.ito.tu-darmstadt.de/publs/cost97.ps.gz>

[McGrew98] McGrew, David A., & Alan T. Sherman, "Key establishment in large dynamic groups using one-way function trees," TIS Report No. 0755, TIS Labs at Network Associates, Inc., Glenwood, MD (May 1998). 13 pages.

[Mittra97] Suvo Mittra, "Iolus: A framework for scalable secure multicasting," Proceedings of the ACM SIGCOMM ’97, 14-18 Sep 1997 Cannes, France.

[Naor98] Moni Naor & Benny Pinkas (Weizmann Inst of Sci, Rehovot), ’’Threshold Traitor Tracing’’, CRYPTO ’98. <URL:http://www.wisdom.weizmann.ac.il/~bennyp/PAPERS/ttt.ps>

[NIST_Sha-1] FIPS Publication 180-1, Secure hash standard, NIST, U.S. Department of Commerce, Washington, D.C. (April 1995).

[Perlman] Radia Perlman, observation concerning LKH [McGrew98] from the conference floor - termed "LKH+"

[Schneier96] Schneier B, "Applied cryptography", 2nd Edition, John Wiley & Sons (1996).

[Shoup96] V. Shoup and A.D. Rubin, "Session Key Distribution Using Smart Cards", in Proc. of Eurocrypt’96 (1996), <URL:http://www.cs.nyu.edu/cgi-bin/cgiwrap/~rubin/keydist.ps>

[Wallner97] Wallner, Debby M., Eric J. Harder, and Ryan C. Agee, "Key management for multicast: Issues and architectures," IETF Internet Draft (work in progress) (15 Sep 1998) draft-wallner-key-arch-01.txt

---

Notes

[i] The watercasting approach that is discussed later challenges this assumption.