Multicast security and its extension to a mobile environment

Li Gong and Nachum Shacham

SRI International, Computer Science Laboratory, 333 Ravenswood Avenue, Menlo Park, California 94025, USA

Abstract. Multicast is rapidly becoming an important mode of communication and a good platform for building group-oriented services. To be used for trusted communication, however, current multicast schemes must be supplemented by mechanisms for protecting traffic, controlling participation, and restricting access of unauthorized users to data exchanged by the participants. In this paper, we consider fundamental security issues in building a trusted multicast facility. We discuss techniques for group-based data encryption, authentication of participants, and preventing unauthorized transmissions and receptions. We also describe the application of these principles and techniques in designing an architecture for secure multicast in a mobile environment.

1. Introduction

A multitude of presently emerging technologies will enable real-time information exchange among multiple participants. Low-cost, high-performance desktop video and audio processing equipment, such as cameras, coders, frame grabbers, and mixers, will make it economical to add real-time generation and display to general-purpose personal workstations. High-speed transmission and switching, now being deployed across the country, will enable instant dissemination of broadband streams over a wide area. Powerful and flexible *multicast* protocols, for dissemination of information to many recipients, will provide the glue that ties the broadband information generation and processing elements with the underlying broadband networks.

A new generation of distributed applications will take advantage of these technologies and provide a quantum leap in network-based services. Multimedia teleconferencing, computer-supported collaborative work, remote consultation and diagnosis systems for medical applications, contract negotiation, and distributed interactive simulation are just some of these applications that require multiparty exchange of data, voice, and video among a large number of simulated and real participants.

Many of these applications will disseminate and exchange sensitive and/or classified information and will require security provisions for session management and information transmission. However, whereas secure communications between two parties (*unicast*) is well understood, little concrete consideration has been given to the security and privacy issues brought about by multiparty information exchange (*multicast*).

Traditionally, teleconferencing has been conducted over private networks or dial-up lines, which provided some physical security measures. However, in integrated networks like the Internet or Asynchronous Transfer Mode networks (ATM) [10], all traffic, including multicast, shares common network resources. Such integration allows a quick setup and flexible maintenance of multicast sessions, but it also brings about serious security and trust issues.

Present multicast protocols, such as IP-multicast, provide the users with little control over who participates in a session, the extent of participation of each user, or even to which users the messages will be delivered [11,2]. There is no security control as to who can register a (dynamic) session address, who can or cannot join a session, who can receive traffic from or inject traffic into the session, or how to deal with the security implications when a host leaves a session. Furthermore, no security management is specified for the multicast hosts or routers. Several ad hoc fixes have been recently proposed, but no systematic security architecture design is available.

Some of this openness was done by choice to facilitate large-scale distribution of unrestricted data, as the successful deployment of IP-multicast demonstrates. However, many multicast sessions in business, commerce, military and medicine are not feasible in such an open environment, and will require controlled participation and data privacy, often with different policies.

A trusted multicast architecture (with associated procedures) is needed to handle these security issues in a coherent manner and to protect network services and applications. Careful investigation of such an architecture can save duplicated, and quite possibly unsophisticated, efforts of inserting security technology into individual applications.

In our paper, we first give a brief review of multicast. We then consider fundamental security issues in building a trusted multicast facility. We discuss new security challenges in multicast and currently available solutions for them. We also describe an initial architectural design for secure multicast in a mobile environment. We conclude

with an overview of related prior work and directions for future work.

2. Overview of multicast

Traditionally, audio or video group communication was conducted over a broadcast medium (e.g., entertainment broadcast, citizens band (CB) or tactical radio), in an open fashion. For example, listening to television broadcast or CB transmissions requires only reception equipment and does not necessitate a prior exchange with the transmitting station. More recently, telephone-based teleconferencing was offered as an extension of pairwise voice conversation, and it requires the establishment of a line between each participating site and a central switch. All traffic flows to the central switch, which replicates the traffic on all lines. Manual procedures are used for authenticating participants, establishing circuits between the remote and central sites, and controlling the flow of traffic.

In networked packet or cell switching (e.g., Internet or ATM), the ability of network switches to replicate user traffic is utilized to construct multicast *trees*, which are more efficient than the star-shaped distribution in phone teleconferencing. Traffic is injected at the root, and each switch on the tree receives the traffic from its parent switch and replicates it on all the links leading to its children switches.

Multicast group addresses are used to identify specific multicast sessions, and network switches maintain tables that relate each group address to a specific set of outgoing links on which packets carrying that address are to be forwarded. To send data to a multicast group, a node sends a single copy of the data addressed to that group. To add a participant to the distribution, the corresponding tree is extended by adding a path from a node on the tree to the new member.

Several multicast schemes have been proposed and are at different stages of implementation and usage. These schemes differ in the ways they construct and maintain distribution trees. The most commonly used scheme is IP-multicast, which is being rapidly deployed throughout the Internet. IP-multicast relies on a subset of the Internet routers and hosts that can participate in IP-multicast sessions. Over this subset, called *M-bone*, all multicast trees are established. When an IP-multicast session begins, a tree is constructed and maintained for each active source. Portions of the tree that do not connect group members are subsequently "pruned".

Under IP-multicast, membership is controlled by the individual users. That is, when a user decides to join a group, the underlying control protocol establishes a path that connects the new member to the tree. This procedure does not involve the sources, who are generally not even aware of who the group members are. Thus, IP-multicast does not provide control over who receives a given multicast transmission. Furthermore, any host can inject packets to any particular IP-multicast group simply by sending packets addressed to the group, even if the host is not a member of the group.

Another Internet-based multicast protocol is Stream Traffic version 2 (ST-II) [41], in which the distribution tree is made of virtual circuits, as opposed to IP-multicast, which relies on datagram forwarding along paths defined by the routing table entries of the time. ST-II originally allowed only the source to initiate the inclusion of a participant in the multicast. Recent improvements, however, provide the means for destination-initiated join, in which the control messages that result in the inclusion of a receiver do not necessarily go to the source [13]. Three schemes for source control of a session population are currently defined:

1. No control. The signaling message arrives only at the tree, and a path is extended to the requester without the source's knowledge.
2. Source notification. While the path is extended, a message regarding the new join is delivered to the source.
3. Source approval. When the join request arrives at the tree, the node that has to begin the path extension process first sends a message to the source, and only after the source approves the join is the path extended.

Both ST-II and IP-multicast employ a distribution tree for every active source in the session. ST-II is used extensively over the Defense Simulation Internet (DSI), however, IP-multicast is in the process of being adopted for the DSI.

A third Internet-based method, named Core Based Tree (CBT) [2], selects a single node in the network to serve as a core. One bidirectional distribution tree is built from the core, and all traffic is forwarded on that tree. We are not aware of any practical experience with CBT. CBT does not specify who controls the tree or how new joins are being made.

Multicast features have also been added recently to ATM technology. The ATM-Forum's User-Network Interface [17] specifies a signaling procedure by which a source can establish a multipoint virtual circuit. Recent proposals extend this mechanism to provide receiver-initiated join. A more advanced ATM signaling technique was designed and implemented, where signaling supports distribution of a hierarchically encoded stream and enables the establishment of multi-VC (virtual circuit), multipoint connections [38].

3. Security challenges in multicast

The various multicast protocols all suffer a degree of

vulnerability to actions that threaten data confidentiality and integrity. Whereas such issues exist also in unicast communications, multicast poses new and unique challenges.

In multicast, just as in unicast, an adversary (who can be a legitimate network user) may eavesdrop on confidential communication, disrupt or distort a session's data exchange, inject unauthorized or bogus traffic, block a session's progress, masquerade as someone else to join in a session, or initiate and operate a bogus session. Therefore, the privacy, integrity, availability, and authenticity of a multicast service must be protected.

However, the nature of multicast presents new problems that cannot be effectively dealt with using trivial extensions of techniques for secure unicast. For example, in setting up a secure point-to-point communication channel, one knows the identity of the party at the other end. In a multicast session, knowledge of the session membership is typically not guaranteed, and session membership cannot be controlled. Another problem is group-oriented secure data exchange. Although using $O(n^2)$ end-to-end secure channels can provide secure group communication, such an architecture loses all the advantages a multicast facility has over unicast. Also, in group-oriented communication, an additional mechanism is needed to reliably establish the identity of the originator of a message. Moreover, group-oriented authentication (and key distribution) is not necessarily $O(n^2)$ pairwise authentication, because of the many possible interpretations of the meaning of belonging to a multicast session.

Mechanisms for multicast security must therefore be developed to regulate participants' access to the data flowing on the tree and their ability to modify traffic already on the tree, protect the transmitted data, and verify to any user the nature of the session in which he or she participated. New protocols must be designed that perform security functions for controlling session organization and management, secure broadcast, and user access to the network.

Multicast security mechanisms can function at the session, presentation, and network layers of the network architecture, where they consist of authentication, encryption, and physical access to the tree, respectively. Techniques for these mechanisms are discussed in the rest of this paper.

4. Trusted multicast

The architecture design of multicast security should consider the placement of the many control functions at appropriate network protocol layers and appropriate network sites, and the trust relationship between network sites and hosts (some of which may function as multicast management centers). Apart from satisfying the security requirements mentioned earlier, a desirable design should also be

- *compatible* with existing network protocols,
- *scalable* to the scope of the global Internet, and
- *transparent* to higher-level applications and services.

Transparency keeps high level applications free from concerns for details of, for example, authentication and the checking of credentials and policies. In addition, the security mechanisms residing at the session and presentation levels should place no restriction on the underlying networking mechanisms. As such they must coexist with networking standards and must not depend on modification of transport and switching elements in the (inter)network. Moreover, the architectural design should be *localizable* in that an area of a network can install the trusted multicast facility and achieve firewall-style self-protection even if other portions of the (inter)-network do not yet support trusted multicast. This feature is important to facilitate a gradual introduction of the new technology to the existing millions of host machines. Finally, the architecture has to be *flexible* enough to support a variety of policies of session control as required by higher-level applications. These features together will make trusted multicast easier to integrate into an existing environment such as the Internet.

4.1. Authentication

Authentication – a process by which one satisfies another about one's claim of identity – is an essential mechanism in trusted multicast to control the registration of a multicast address and screen hosts that ask to join an existing session. Often, the requesting party *and* the existing session members need to authenticate each other. Sometimes one party may delegate some of its authority to another party. This can be achieved by letting the former issue a credential, such as a signed and verifiable certificate, to the latter for later presentation together with a proof of identity.

A very rich body of literature exists on the topic of authentication and delegation in distributed systems (see [1] for references). Thus, we concentrate only on the extension of the basic pairwise authentication model to the internal and external authentication of all existing members of a multicast session as a group.

There are two distinct issues here. One is who controls the membership of a group and how to enforce such a membership policy. The other is what constitutes being admitted to an existing group. We discuss the second issue first.

Except in the case of anonymous sessions, a minimum requirement for belonging to a group is that the new party should be introduced to (some or all) existing members. This introduction merely announces the iden-

tity (i.e., the name) of the new party, and can be done by any member, by at least some preset number of members, or by a group leader. Whoever handles the introduction process in essence implements the admission policy. A more stringent requirement can be that each existing member and the new member must successfully authenticate each other after an initial introduction. An even more strict requirement can be that every member must also know that every other member knows about the new party.

To require pairwise authentication, $O(n^2)$ runs of an authentication protocol are needed for a group of size n, although such runs are done incrementally as new members join the group. A more efficient method is to elect a group leader who is trusted (by other group members) to properly authenticate the new member. In this case, the group leader in effect controls the group membership policy. A natural choice of the leader is the session initiator. When registering the session, the initiator becomes the leader and specifies an admission policy. If the leader crashes or retires, a new leader must be elected through a secure protocol, possibly according to the session policy. It is conceivable that some members may monitor the leader's actions to check conformance with the policy.

The overall security of the session clearly is related directly to who the participants are. A change of membership may affect the kind of security the session can provide. It may also be the case that a change in the required session security forces a change in the session membership. A member leaving a session, especially a long-running session, voluntarily or otherwise, should properly sign off and erase any sensitive information related to the session, such as the group key. Otherwise, this member can later eavesdrop without explicitly joining the group, or may leave behind the residue information that can be exploited by potential attackers. Since even a benign member may forget these procedures, the group key should be updated whenever a member leaves a session. For the same reasons, garbage collection must occur so that the group removes any member not doing a proper sign-off. The status of members can be monitored by a number of methods, such as by exchanging "heart-beat" messages.

4.2. Secure sessions

Based on previous experience, both ours and that of others, in the area of secure group-oriented distributed systems (e.g., [35]), we can identify several crucial issues in the forming of secure sessions.

Session membership policies. Sessions can be set up for various purposes with different membership policies. At one end of the spectrum is a totally open policy. Anyone is free to take part in such an open session, which resembles a mass gathering or an unmoderated USENET newsgroup. A more restricted session, much as a presidential town hall meeting, has open participation but members have to pass a security check and the session has a moderator. An even more restricted session is like a board meeting, where only the board members can participate. From time to time non-members are invited, but only if a majority of the members consent. We can easily imagine an infinite number of such policies (including the multilevel security variety [27]). As more restrictions are added, we reach an ultra-secure and closed session. As we argued earlier, the multicast architecture should be flexible enough to accommodate many policies so that each individual application or each session can choose to implement one policy or a set of basic policies.

In some applications, membership information itself is sensitive; thus, rules must be defined for disclosing the current group membership. However, traffic analysis may still reveal membership. One solution is to have trusted gateways that mask all sources and destinations. Another solution is to saturate the network so that an observer cannot distinguish between a meaningful packet and a junk one.

Registration and deregistration. When a host or user requests a network address to establish a multicast session, a multicast management center (MMC) must verify the requester's identity and check the session policy dictating who can register a session. Similarly, the requester may also want to verify the identity of the MMC to be sure that the session is correctly registered.

Information such as membership policy, user credentials, and records of registered sessions can be stored at the MMC in the form of an access control list.

The MMC can be a centralized service or a distributed one. When a network has more than one MMC, all MMCs must coordinate among themselves to ensure consistency and uniqueness of multicast addresses. They may or may not adopt an identical admission policy.

When a request to de-register a session arrives at an MMC, it must authenticate the requester and determine if it is authorized for such a request. Normally, only the session leader, or its representative, can de-register the session. When the session leader crashes or when it forgets to de-register, a "dead session" must be garbage collected. This can be done by having the MMCs periodically check the activities of all registered sessions. Similarly, if a multicast session is deemed undesirable (e.g., when it floods the network with junk), the session must be terminated.

Joining and leaving a session. The communications among session members can be public or private. This can be specified at registration time, and can be changed later. It is also possible to leave this attribute to the discretion of the individual members, so that some communications may be private while the rest remain public.

For private sessions, adequate privacy provisions are required in multicast. Because most networks are open in the sense that anyone may eavesdrop on network traffic, we need to encrypt message contents when requested to ensure that only those authorized can correctly decrypt them.

Secure session communication. The traditional approach is to arrange the distribution of a common encryption key, shared by all session members, and encrypt broadcast messages with this group key.

With this approach, an initial key distribution must take place when a session is registered. This key can be chosen by either the session leader or the authentication server. The key is then maintained by the session leader or stored at the MMC where the session is registered. When someone applies to join a session, the group leader or the MMC checks the session policy to make a decision. When a new member is accepted into the session, it receives the group key from the MMC or the group leader. When the group leader crashes, a new leader is elected, using any of the many known election protocols.

A significant shortcoming of this method is that, if session members can be dishonest, the group key does not identify the source of a message. This issue can be solved by requiring that each member sign its message using a digital signature technique [37]. Another problem is that frequent key changes (and their synchronization among group members) during a busy session affects performance. Secure broadcast algorithms alleviate these two issues, as we explain below.

Secure and efficient broadcast. When a member can choose a new encryption key for a new membership on a per-message basis, the risk of key leakage by a departing member is considerably reduced and a new group key is unnecessary.

Suppose each member has a pair of public and private keys, for example, for use in the RSA cryptosystem [37]. Each member must have knowledge of the group membership – at least he must know the public keys of all the other members. To broadcast a message, a member chooses an encryption key (for use in DES [42]) at random and encrypts the message. The member then signs a timestamp (to defend against replay attacks), the DES key, and a one-way hash function [32] of the original message (to serve as an integrity checksum), using its private key. The member then encrypts this signature with each other member's public key. Finally, the member broadcasts (or multicasts) the $(n-1)$ encrypted signatures and the encrypted message. (Here, n is the size of the group.) At the receiving end, a group member decrypts a suitably encrypted signature (with its private key), obtains the DES key (using the broadcaster's public key), and then retrieves the message. Clearly, anyone who is not a group member would not be able to retrieve the message. This simple algorithm has the following attractive features: (1) the encryption key can be changed for every message; thus a membership change does not require additional security measures such as changing the group key and (2) only the message header (containing the encrypted signatures), and not the message body, increases linearly in length with the number of destinations. Moreover, the computation of a one-way hash function can be very efficient, and the checksum adds only a few extra bytes [32].

To use this broadcast primitive, session members receive and cache others' public keys and other relevant data. When notified of session membership changes, members simply update their local membership lists and choose new DES keys for future multicast, which are thus made unreadable to those outside the session (including the members who just left).

If group members do not have public key capability, the above algorithm cannot be easily extended since it appears at first sight that in addition to every pair of members needing to share a distinct key (thus brings about the $O(n^2)$ key problem, which however can be alleviated to some extent [21]), the message body needs to be encrypted $(n-1)$ times (e.g., using DES) and thus the broadcast message body also increases with the size of the group, making the algorithm less scalable. Fortunately, there exist secure broadcast algorithms with the same attractive features of the one using public-key cryptosystems. We will discuss them in section 4.3.

Encryption mode. It is generally undesirable to use the basic Electronic Code Book (ECB) mode [42] because each block is independently encrypted and, thus, the mode is more vulnerable to slicing attacks (e.g., by reordering blocks) and cryptanalysis.

In the plaintext-feedback mode, the encryption of a block depends on the plaintext of the preceding block(s). This results in infinite error propagation in that a block cannot be correctly decrypted unless all preceding blocks are correctly decrypted. In other words, a single lost packet (or cell) or any data corruption during transit renders the rest of a message unreadable. This undesirable property may necessitate excessive retransmission when the network or a stream is unreliable.

The ciphertext-feedback mode, on the other hand, has a limited and controllable error propagation, typically of two encryption blocks. As long as two consecutive ciphertext blocks are received correctly, decryption can proceed and resynchronization is automatic. Moreover, in some applications involving audio and video, occasional packet loss does not matter much because it causes only minor degradation in voice or video quality. These advantages make ciphertext-feedback the favored encryption mode.

The multicast facility may need to adapt between multiple encryption modes, and between using block ciphers and using stream ciphers, according to the application environment. For example, the ECB mode allows

random access to individual blocks of a message. In addition, a stream cipher, especially if precomputation of the stream is possible (e.g., using DES as a pseudo-random number generator), is faster than a typical block cipher, but because of the cost of resynchronization when data loss occurs, it is more suitable in a highly reliable environment.

Finally, we consider the case of partial encryption of streams. One motivation to avoid encrypting the whole stream is efficiency – for example, when it is possible to identify crucial data (such as control information) that are the only portion (of the stream) to be encrypted for security. The sending and the receiving ends must be in synchrony as to which portion is encrypted and which is not.

Another motivation for partial encryption is that not all receiving ends require the same amount or type of information from the source. One member may want to hear the voice more clearly, while another wants to improve the image quality.

Similarly, different portions of the same stream may need to be encrypted differently. For example, the background of a picture may be public knowledge, but the human image in the center may be highly classified and need special treatment.

In such cases, it is conceptually simpler to view partial streams as many layers of the same stream, and techniques developed for Heterogeneous Multicast (HMC) [38] can be used to explore the concept of multicast sessions in which users participate at different bandwidth levels.

Session advertising. Many services need to be advertised. The existence of a registered session can be announced – for example, by a bulletin board service. Proper credentials and identification information must be provided so that one can verify the authenticity of the advertisement. Sometimes it is necessary to restrict the dissemination of service information. For example, the bulletin board can be made multilevel secure.

4.3. Secure broadcast and common attacks

When S must send a message m to a number of destinations $P_i, i = 1, \ldots, n$ (for example, a subset of nodes on a network) and the message content must be kept private for the intended recipients, the message is usually encrypted before being broadcast. Depending on the pattern of key sharing among communicating parties, different protocols can be used. For example, S can encrypt m with k_i and send the ciphertext to P_i. This amounts to a total of n encryptions, and n point-to-point messages or a broadcast of n pieces of ciphertext. If S and the P_i's share a group key, one encryption and one broadcast of a single ciphertext is sufficient. However, S may need to send messages to different sets of recipients and thus must maintain many such group keys, up to the order of $O(2^n)$. The problem we examine here is how to securely broadcast a message when public-key systems are not available, and how to avoid encrypting the message n times or maintaining $O(2^n)$ group keys.

Some authors of secure broadcast protocols (e.g., [9,3]) focus their attention mostly on attacks on secrecy, and do not discuss other attacks, including the insider attack we describe later in this section. We highlight the risks with a case study of the Chiou-Chen protocol [9] where we discuss these attacks and suggest countermeasures.

We use the following notation. Plaintext x encrypted under key k is denoted as $\{x\}_k$, and the concatenation of x and y is (x, y). The Chiou-Chen protocol works as follows. S selects a random encryption key k, encrypts m with it, and computes $\{k\}_k$. Then S solves the equation

$$x \equiv \{k\}_{k_i} \bmod n_i \tag{1}$$

for x using the Chinese Remainder Theorem. Here, k_i is P_i's public key. n_i is a publicly known number associated with P_i, and these numbers must satisfy a few conditions, including that they be pair-wise prime. (See [9] for details.) Then S broadcasts a message of the form $(S, x, \{k\}_k, \{m\}_k)$.

After receiving the broadcast, P_i computes $\{k\}_{k_i}$ (from x and n_i) and decrypts it with k_i to obtain k, with which P_i decrypts $\{k\}_k$. If the last decryption does not reveal k, it means either that the message has been modified during transmission or that the broadcast is not intended for P_i. Otherwise, P_i uses k to decrypt $\{m\}_k$, thus obtaining m. Note that m is encrypted only once, and only one piece of ciphertext is broadcast. The integrity of $\{m\}_k$ must be checked separately.

The protocol implicitly requires strong (and possibly impractical) assumptions of the protocol execution environment. When these assumptions do not hold, the protocol is vulnerable to replay and masquerading attacks, especially by a legitimate but malicious recipient of a broadcast message. For example, in the original protocol, anyone can record S's message and play it back again. A recipient cannot detect a replay except by keeping a record of past messages and keys, thus noting message repetitions. If recipients cannot be expected to keep such records, establishing the freshness of a message by other means is essential. For the sake of discussion, we assume that clocks are securely synchronized. (Other methods to ensure freshness, such as challenge and response, can also be used.) In this case, a usual solution is to attach to m a timestamp t to indicate the time of message creation – that is, replacing m with (t, m) in the protocol.

However, an insider (say P_1) can still attack. Being a legitimate recipient, P_1 knows k after receiving the broadcast. Thus P_1 can replace (t, m) with (t', m'), where t' is a current timestamp and m' is an arbitrary message, and then resend the message. If other recipients cannot detect that k is reused and there is no other hardware or

software mechanism to identify the message origin, the new broadcast message will be taken as from S rather than from P_1, with potentially serious consequences. Timestamping $\{k\}_k$ by broadcasting $(S, x, \{t,k\}_k, \{m\}_k)$ still has the same weakness because P_1 who knows k can easily forge a seemingly legitimate $\{t',k\}_k$ with a new timestamp t'. The victims are confined to recipients of the original broadcast.

One solution to these problems is to timestamp key k by modifying the equations to be $x \equiv \{t,k\}_{k_i} \bmod n_i$. This is clearly better because t can specify k's creation time as well as its lifetime. Since P_1 does not know k_i, $i \neq 1$, P_1 cannot forge $\{t,k\}_{k_i}$. Thus P_1 cannot convince others that an expired key k is still valid, and a future message broadcast by P_1 using k will be rejected.

There is yet another problem with this new version. In many systems, such as the Internet, a broadcast channel is a simulation that will behave correctly only if all parties concerned follow the protocol (e.g., [4]). In such an environment, it is likely that P_1 can interfere with the broadcast before another recipient receives the message. In particular, P_1 can intercept the broadcast message m from S, change its content to m', and then send it on to other recipients before k expires. This type of attack is often possible when the message delivery system is unreliable or slow, and especially when broadcast is simulated in software instead of implemented in hardware.

One protection against this attack is to omit the intermediate key k altogether, solve $x \equiv \{(t,m)\}_{k_i} \bmod n_i$, and broadcast (S,x). This change makes the protocol secure against described replay or masquerading attacks by outsiders as well as by insiders, since no one besides S and P_i can forge $\{(t,m)\}_{k_i}$. (Note that $\{(t,m)\}_{k_i}$ can be broken into smaller blocks if necessary.) The efficiency of this version is not as good as the original Chiou-Chen protocol, however, because m, which may be significantly longer than k, is now encrypted n times. Nevertheless, only one piece of ciphertext is broadcast, and its length stays roughly the same as in the original protocol.

The new version can also be slightly modified to broadcast *different* private messages to multiple destinations. For example, suppose S wants to send m_i privately to P_i. Instead of sending n separate messages, S can solve equations $x \equiv \{t,m_i\}_{k_i} \bmod n_i$ for x, and then simply broadcast x.

Suppose that some recipients cannot store past messages, and also suppose that there is no hardware or software support to ensure that a broadcast cannot be interfered with before the message reaches all destinations. To consider insider attacks under these conditions, we propose a new protocol that achieves the level of efficiency of the original protocol. Suppose $h(\,)$ is a one-way hash function. The sender S selects k and encrypts m to obtain $\{m\}_k$. S then solves the set of equations $x \equiv \{t,k,h(m)\}_{k_i} \bmod n_i$ for x, and broadcast $(S, x, \{m\}_k)$.

Here $h(m)$ acts as a checksum of m. Recipient P_i obtains m as before, computes $h(m)$, and compares it with the checksum value received from S. A discrepancy suggests accidental modification or deliberate tampering of the broadcast message. This protocol's efficiency is close to that of the original protocol because m is encrypted once, only one piece of ciphertext is broadcast, the computation of a one-way hash function can be very efficient, and the checksum adds only a few extra bytes.

Since the original Chiou-Chen protocol is secure against known-plaintext attacks [9], and the new protocol is different only in that an extra timestamp t and a checksum $h(m)$ are encrypted, the new protocol is also likely to be resistant to known-plaintext attacks [9]. Moreover, since the attacker does not know k_i and thus cannot modify $\{t,k,h(m)\}_{k_i}$ to change the checksum, any illegal modification of the broadcast message will be detected by P_i.

If $h(\,)$ is a secure keyed one-way hash function, then the amount of encryption and the length of the broadcast message can be further reduced by replacing $\{t,k,h(m)\}_{k_i}$ with $(t, h(k_i,m,t))$ in the equations and broadcasting (S,x,m,t). After obtaining m by decrypting $\{m\}_{k_i}$, P_i can recompute $h(k_i,m,t)$ and compare it with the received value to check message integrity. In fact, if the privacy of the broadcast message is unimportant, no encryption is needed and the protocol can be further simplified.

4.4. Secure broadcast using polynomial interpolation

We recently proposed the NED techniques with which a sender can distribute a session key to n recipients [19]. Thus, intuitively, the sender can also broadcast a (secret) message to the n recipients using the same NED method and replacing the session key k with a message m. In such a broadcast, unlike in authentication, a recipient may not need to know exactly all the identities of other recipients. When m is long, it can be broken into smaller blocks. We call such protocols NED-B, where B is for broadcast [19].

The notation we use in this section is as follows. S is the sender, who wants to broadcast a message m to n recipients, whose identities are $I_1, I_2, \ldots, I_n$. Each recipient I_i shares a secret key K_i with the sender. $x \mid y$ denotes the concatenation of strings x and y. Function $h()$ is a keyed one-way hash function, where the first argument is the secret key, and the rest of the arguments are concatenated to form an input string. The output from the hash function is of a fixed size, independent of the size of the input string. T_s is the timestamp of sender S's clock. r is a random number that S chooses each time a message is to be broadcast.

After deciding on m and choosing r, the sender solves the following equations to obtain values of $a_1, a_2, \ldots, a_n$.

$$\begin{cases} m \mid m + a_1 \times h(k_1, I_1, r) + \ldots + a_n \times (h(k_1, I_1, r))^n \\ \quad = h(k_1, I_1, T_s, r, 1), \\ m \mid m + a_1 \times h(k_2, I_2, r) + \ldots + a_n \times (h(k_2, I_2, r))^n \\ \quad = h(k_2, I_2, T_s, r, 2), \\ \ldots = \ldots \\ m \mid m + a_1 \times h(k_n, I_n, r) + \ldots + a_n \times (h(k_n, I_n, r))^n \\ \quad = h(k_n, I_n, T_s, r, n). \end{cases} \tag{2}$$

Let $f(x) = k + a_1 \times x + \ldots + a_n \times (x)^n$. The sender then broadcasts $(S, r, T_s, f(1), f(2), \ldots, f(n))$.

Recipient P_i can use interpolation to retrieve m from the following equations:

$$\begin{cases} m \mid m + a_1 \times h(k_i, I_i, r) + \ldots + a_n \times (h(k_i, I_i, r))^n \\ \quad = h(k_i, I_i, T_s, r, i), \\ m \mid m + a_1 \times 1 + \ldots + a_n \times 1^n = f(1), \\ m \mid m + a_1 \times 2 + \ldots + a_n \times 2^n = f(2), \\ \ldots = \ldots \\ m \mid m + a_1 \times n + \ldots + a_n \times n^n = f(n). \end{cases} \tag{3}$$

Note that for the above equations to have a solution, we require that $h(k_i, I_i, r)$ be out of the range of 1 to n. If this is not satisfied, the sender can select a different value of r.

P_i also determines whether in the result value the first half is identical to the second half. Any modification to the sender's message will cause (with a high probability) the retrieved value to be not of the form $m \mid m$. This is because the attacker does not know $h(k_i, I_i, r)$, the value for which P_i computes $f()$, and thus cannot modify k in any predictable fashion by modifying the $f(i)$'s. Obviously, the number r should not be reused. Suppose r is first used to broadcast more than one message where P_i is a recipient; then, P_i is able to compute $h(k_j, I_j, r)$ and $h(k_i, I_i, T_s, r, i)$, and thus can forge future messages.

Moreover, to detect replay attacks, the sender must include a freshness identifier in his message, so that each recipient can be convinced that the message is indeed fresh. A freshness identifier can be the sender's timestamp, if the sender and the recipients have securely and closely synchronized clocks. It can also be a nonce, such as a random number, generated by the intended recipient. The freshness identifier must be securely bound to the message in such a way that an attacker cannot modify the identifier without being detected. The computation of $h(k_i, I_i, I_2, T_s, r, i)$ assures P_i that the message comes from S and is fresh.

An outside attacker who monitors network traffic cannot compute m because he does not know any of the k_i's and thus cannot compute $h(k_i, I_i, T_s, r, i)$. Although an insider P_j can compute $h(k_i, I_i, T_s, r, i)$, this value is used as a one-time pad and cannot lead to the compromise of k_i. Note that, even though the argument here is similar to that used in Shamir's secret-sharing scheme [39], the secrecy argument against insider attacks is no longer information-theoretic. This is because P_j can guess a value of k_i (call it k_i') and verify the guess by computing $h(k_i', I_i, T_s, r, i)$ and then comparing it with $h(k_i, I_i, T_s, r, i)$. A match indicates that $k_i = k_i'$ with a high probability.

However, if we assume that the selection of k_i is random and from a sufficiently large space, then such an exhaustive attack is computationally infeasible. In an exhaustive cryptanalysis, the attacker may attempt to build up a dictionary of possible session keys and hash values [14]. One way to increase the difficulty of this attack is to use longer texts in the input to the hash function. This method is likely to greatly increase the search space with little loss of efficiency, because the computation of a hash function is extremely fast and the length of the hash value remains constant.

Since the size of q (in mod(q)) is $O(\log n)$, given a message m of $O(\log n)$ bits, the overall length of the broadcast message has $O(n \log n)$ bits. In an environment where the recipients are scattered throughout a large network, the savings in terms of efficiency by avoiding conventional encryption may be outweighed by the increased total network traffic and load. In this case, the sender can use the NED method to distribute just an encryption key, and then broadcast the whole message encrypted with this key. Such a key can be changed on a per message basis; or it can be changed only when session membership changes. This arrangement is quite similar to the algorithm in section 4.2 that uses public key cryptosystems.

Finally, we point out that the use of a one-way (hash) function (or another similar mechanism) cannot be avoided in our setting where each recipient holds (i.e., shares with the sender) only one secret that is long-term (i.e., does not or cannot be changed frequently between authentication sessions) and small in size (e.g., 8 characters), and the session key is of a similarly small size. This is because that, without the one-way hash function, A's finding out B's key k_2 is just a matter of solving a higher-order equation. This intuition is supported by the work of Blundo et al. [5], who proved a theorem (for the non-interactive case of conference key distribution) that to prevent a group of colluders from compromising others' secrets, a user needs to hold secret information that is many times the size of the session key. The authors also give a protocol using symmetric polynomials (but not one-way hash functions). This intuition that one-way function is necessary for third-party-based key distribution has been recently proved in [18].

Now we briefly discuss related work in secure broadcast. A number of protocols exist for secure broadcast, and some protocols for conference key distribution can also be extended for this purpose (e.g., [9,26,3]). The three protocols cited here are typical in that they do not consider attacks except those directly on message secrecy, as we can see from section 4.3. Among these pro-

tocols, the closest to the our NED-B protocol is perhaps the work by Berkovits [3], whose proposal uses polynomial interpolation but in a slightly different form.

The Chiou-Chen [9] protocol is more expensive because it needs to associate one positive number with each of the n recipients where the numbers must be relatively prime to each other. Managing the allocation of such numbers to a large group of recipients – for example, hosts and users on the Internet – can be a serious problem in practice. Moreover, the arithmetic must be computed in a field containing the product of all the n relatively prime numbers. Since the range containing n prime numbers is roughly $[1, n \log n]$ [22, pp.9–10], the size of the field is at least $O(n \log(n \log n)) = O(n \log n)$ bits long and the broadcast message has $O(n^2 \log n)$ bits, whereas in the NED-B protocol, the field size is only $O(\log n)$ and the broadcast message has $O(n \log n)$ bits.

4.5. Control of tree access

Authentication guarantees that only certified users become legitimate participants, and group-oriented encryption enables only those participants to decode the data being sent. Yet, the current network-level multicast routing and forwarding mechanisms allow any user to establish a path to the tree and to inject traffic at will. It is thus of interest to employ network-level mechanisms that limit physical access to the distribution tree.

A simple approach, used by ST-II [41], is to require all tree attachments to be initiated by the source of the tree. This makes the source aware of who is listening and allows it to reject unwanted requests. If the source approves a connection request, it sends a message down the tree toward the destination. At one point, the message leaves the tree and establishes an extension path while progressing toward the destination. While this technique provides some level of control, it adds significant processing load to the source, thereby limiting the session size.

In a teleconference, receivers usually initiate attachment to a specific distribution tree. If the distribution is by multiple source-based trees, a typical receiver is likely to shift attention from source to source, and thereby frequently changes its attachment to the respective source-trees. Even if the distribution is over a single core-based tree [2], a receiver changes its resource allocation on that tree to reflect the shifting emphases on different sources. Both resource allocation and the establishing and disconnecting path involve actions by multicast agents residing at network nodes.

As was argued above, requiring the sources to mediate these actions reduces efficiency and performance. It is thus desirable to allow receivers to interact directly with the multicast agents, with minimal if any involvement by the sources. This receiver-agent interaction must be supported by proper authentication measures that prevent unauthorized users from directing traffic in their direction.

The process of authentication of end hosts to network switches is somewhat different from authentication among peer hosts. First, because of storage and processing limitations, switches cannot be expected to maintain keys, public or private, for all hosts connected to the network. Moreover, at a time a connection is initiated, a host does not know which switches will actually modify their routing tables to support its request. That is, advance key exchange with the "right" switches is not feasible.

Each network switch could maintain a small number of public keys, with the corresponding private keys held at one or more authentication servers. Before initiating a request for a new connection or modification of an existing connection, a host first sends the request message to the authentication server. After authenticating the host, the server sends back the message with a signature attached. That message is sent along the network path, and every switch on the way checks it against the signature, using its locally held public key of the server.

Similarly, a node that wants to inject traffic must first obtain a signature from the server. A network node will not accept traffic from a source without such a signature.

Notice that authentication of receivers is done relatively infrequently and is not likely to impose a great processing burden on the switches. In contrast, to block unauthorized sources, packets must carry a signature, which results in a much heavier load on the switches, especially in the case of stream traffic – for example, video. For stream traffic switches, only a fraction of the packet should carry signatures, and the switches should cache the signatures for forwarding of those packets that do not carry signatures. The cache should be refreshed every so often.

It is also conceivable that malicious or ignorant parties may misuse network resources – for example, by flooding a multicast address, or by setting up a bogus multicast session with the sole purpose of using up network bandwidth. Simply ignoring such "junk" packets (like people throwing away junk mail) is inadequate because by the time such packets reach their destinations, precious network resources have already been consumed. Thus we need to block such usage (as soon as it is identified) as near to the multicast source as possible. We can imagine that the MMCs act as warning posts that send out advisory messages to individual hosts, who in turn regulate local network access. Many Internet sites have already started filtering incoming and outgoing packets, but usually for different reasons. Alternatively, ideas similar to those developed for the Visa protocol [15] can also be used. However, blocking misuse seems to require modifications to the commonly used multicast (and unicast) protocols and software. For example, user processes need to obtain authorization for

sending multicast packets, possibly in the form of certificates, and network gateways have to verify the certificates before granting the necessary network resources.

4.6. Scalability and trade-off

The task of assigning initial encryption keys for user or host authentication grows only linearly with the number of users and hosts interested in participating in secure multicast. The number of messages for establishing a session and for changing the session membership is also linear to the size of the session. A secure broadcast algorithm (such as the NED-B protocol in section 4.4 and in [19]) allows dynamic encryption key change on a per-message basis, thus eliminating the need for a new group key upon group membership change. Software implementations of encryption algorithms are sufficiently fast, while hardware implementations will further enhance performance. In protocol NED-B(n), the most costly activity – the encrypting of (potentially long) broadcast messages – remains constant when the session size increases. Only the message header (which contains instructions for decryption) increases in length, and only linearly, when the session size increases.

Nevertheless, at present, a quality public-key cryptosystem such as RSA typically has a key length of 512 bits. This means that a multicast message header increases 64 bytes for each additional member. Such an increase may not be practical in some situations. One trade-off is to use short RSA keys or use different public key algorithms. Another is to selectively (and dynamically) choose between using a group key and using secure broadcast, depending on the importance of each session and perhaps each message. For example, when group membership becomes reasonably stable, it is more efficient to use a group key.

Sometimes, members of a session also form a large number of (sub)groups. For example, in a teleconferencing application, although everyone's voice is broadcast to all destinations, a member may want to broadcast his physical location information (his coordinates) only to members in his proximity. To structure such (sub)groups, a secure broadcast algorithm has the advantage that, since data encryption keys can be chosen dynamically on a per-message basis, a member need not store a potentially large number of (sub)group keys. Of course, a member can optimize by storing data encryption keys and the corresponding signed message headers and reusing them until a membership change occurs.

4.7. Multilevel secure multicast

End-to-end encryption can be used to control message dissemination on a discretionary basis. Mandatory control, possibly of a multilevel structure [27], can also be implemented. Here, groups and members are classified at different levels, and a party cannot become a member of a group whose security level is higher than its own. Levels are also assigned to encryption keys. A member cannot have access to a key whose level is higher. For encryption, the level of the key must be equal to or higher than that of the plaintext. The level of a ciphertext is then the level of the key. Such an arrangement effectively segregates the multicast network into virtual networks, each at a distinct security level.

Multilevel security may require that lower-level information be accessible to higher-level group members. One way to achieve this is for the lower-level group members to be able to "write up" – to send multicast messages to groups at a higher level. However, since the sender typically cannot have the suitable higher-level encryption key, there must be a trusted multilevel network component that either makes the lower level keys available to the higher level members or acts as a gateway that decrypts an incoming message with a lower-level key, and re-encrypts the message with a higher-level key, before relaying the message. Alternatively, low-level users may write up using a high-level public key.

There can be issues of covert channels, where lower-level members can observe the existence of high-level network activities. One solution is to use relays to confuse observers. For example, if messages coming out of SRI go through a relay that re-encrypts them with a key shared with the IBM relay, then a third party can deduce only that someone at SRI sends a message to someone at IBM. Given a sufficiently random communication pattern between SRI and IBM, the bandwidth of this covert channel can be greatly reduced. The two relays can exchange useless messages at random intervals to further confuse the third party. Although many applications are not threatened by information leakage through covert channels, the same techniques can be used to impede traffic analysis, which is often deemed a threat to privacy.

5. Mobility in trusted multicast

Mobile computing is expected to increase significantly in the near future [24] due to dramatic decrease in weight and power consumption of processing devices, and the availability of new frequency bands and wireless services. Including mobile users in multicast sessions will result in powerful applications that include a "flying doctor" initiating a multiparty consultation session, a business traveler taking part in a meeting being conducted at his company headquarters, and members a rescue team coordinating their operation in real time, all through multicast in (inter)networks such as today's Internet. Data privacy (e.g., data confidentiality) and integrity (e.g., data authenticity) are essential parts of the communication architecture that supports such applications. In this section we outline an architecture for incorporating mobility in trusted multicast. First,

however, we discuss the unique issues of secure multicast in a mobile environment that affect such an architecture.

5.1. Mobility constraints

The constraints of mobile users are due to:

- Host limitations on power consumption, storage, and processing speed.
- Changes in location and network connectivity, which require a user to occasionally communicate through a "foreign" network that cannot be trusted as the "home" network.
- Limitations of wireless channels, including lower capacity and wider broadcast range than their wired counterparts.

Power, storage, and bandwidth limitations restrict the ability of mobile users to participate as peers in the procedures for managing secure multicast groups. The broadcast nature of wireless channels increases the risk to data privacy, and location changes require a mobile host to adapt to the security level offered by the connecting network.

Practicality dictates that we allow a gradual deployment of the security architecture such that the user community can start small and grow gradually while depending on or interfering with other network services or machines as little as possible. It is equally important that the added security mechanisms should coexist with popular security services such as fire-walls. Moreover, the security mechanisms in mobile hosts, such as for key distribution and data encryption, should be kept minimal. To summarize, the security architectural design emphasizes the following features:

1. *transparency* – security features are independent of, and do not affect, the underlying multicast protocols and routing algorithms,
2. *self-sufficiency* – users or hosts wanting to engage in secure multicast can do so without depending on other disinterested users or hosts or demanding that they take additional actions,
3. *mobility* – anyone with access (directly, or indirectly via an agent) to multicast, and with additional facility for security, can initiate and participate in secure multicast sessions, and
4. *simplicity* – little additional software or hardware for security is required in mobile hosts;
5. *efficiency* – to make optimal use of scarce processing and communication resources that are dominant in mobile computing.

5.2. Architecture outline

The architecture discussion consider two cases:

- Accommodating a single mobile host.
- Incorporating a group of hosts that participate in a multicast session through a shared network.

For the first case in which a mobile host connects to a session from a foreign network, our proposed solution is centered on the concept of an agent [28]. In our context, an agent is a host with access to multicast services – for example, it is on the M-bone or a multicast tree – but is also equipped with an additional facility for security. An agent has the following tasks:

- Session management, including membership registration and deregistration and control of resource reservation.
- Authentication and key exchange.

An agent may serve a mobile host at a level of full trust in which the agent carries on behalf of the mobile host all session management and security related functions. In this case, communication and signaling between a mobile host and its agent is by unicast. When a user wants to know about or participate in (secure) multicast sessions, he locates an agent with which he establishes a secure (i.e., private, and authenticated) communication channel. A properly set-up agent should possess the necessary encryption keys for sessions in which the user is authorized to participate. The user sends his input to a session via the agent, who encrypts the traffic with a suitable key and then relays the encrypted text by the underlying (insecure) multicast mechanism. The agent also watches for any traffic, often encrypted, that might be of interest to his client, and decrypts and forwards this traffic to his client (via the secure channel). In other words, an agent's function can include both data encryption and information filtering.

It is also feasible for an agent to operate at lower trust levels, such as partial trust and no trust. An agent operating under partial trust carries out on behalf of the mobile host only session management tasks. However, group encryption keys are held and used by the mobile host and are not known to the agent. An agent that operates at the no-trust level is basically a regular multicast switch that replicates traffic and forwards signaling messages but does not interact with the end host. The trust level between a mobile host and the multicast agents can change during the session. At the beginning of the session the host may use a full-trust agent, possible located at his home network, to obtain and distribute certificates and keys. Once the mobile host is well established in the session it can take responsibility for data privacy and assign another agent to forward the multicast messages on a shorter path that his home agent.

Figure 1 depicts a mobile host (M) that connects to a multicast distribution tree via a wireless broadcast channel. Initially M uses a host (H) in its home network as an agent, at the full trust level. H participates in the secure multicast session and forwards data to and from M via the tunnel they established. To shorten the data path to M, a new agent, F, is assigned on a foreign network. F still participates in the session, but forwards encrypted data only. Finally, a router (R2) on the network on which M resides is assigned as a no trust agent.

The advantages of the agent-based approach are simplicity and backward compatibility. For example, security issues are separated from transport and routing issues. Thus, the super-imposed security mechanism can coexist with, and utilize advances in, networking technology without needing major change. Also, the burden of providing security is on the mobile hosts and the security agents, and not on disinterested machines and routers. Moreover, a mobile host with very limited processing power can request that its agent (which can be better equipped) perform most of the CPU/disk intensive task. To reduce bandwidth consumption, the agent can also perform most of the data filtering and organization work for the mobile host. These features are quite distinct from those typically suggested for securing mobile IP (see [16] for some references), although those proposals also use concepts such as agents and subscription.

5.3. Locating an agent

An agent can be a host dedicated to this function; or a host can become an agent upon request. In a relatively static setting – for example, the users being staff of a corporation and using machines in their offices – an agent can be a server in a local-area network. Depending on the security condition within a local environment, the channel between an agent and a client can be either encrypted or open. For example, within the headquarters of the corporation, a channel can be a wired cable or can be through a plain Ethernet. However, if certain sessions are sensitive even within the corporation, then certain traffic must be encrypted. Various performance optimizations are possible, such as using a stream cipher and pre-computing the encryption key stream.

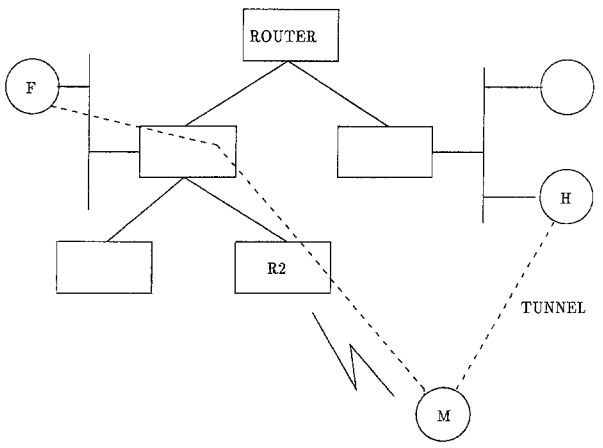


Fig. 1. Mobile host and multicast agent.

In a more mobile setting – for example, when a traveling executive takes part in a session from a hotel room or some other public network connection point – one can only place a minimum trust on alien hardware and software. Thus an agent is typically located within the executive's laptop or hand-held device, which in this case acts as a fully functional node on the M-bone. Note that the mobile computing device need not retain its home IP address, and may not need an "in-care-of address" either, as long as it can participate in multicast and obtain access to session information.

Alternatively, the user must locate an agent somewhere in the network, and can use point-to-point messages to inquire about the existence of a suitable agent nearby. For example, he can contact his agent at home for references. The user can also multicast a message similar to that used in a local-area network to locate the network file server, in which case the user must be equipped with direct access to multicast. Such a message can actually be an "anycast" message [30] to a well-known address (e.g., the telephone number for directory service) that should be the same no matter where the user is in a country even though in different areas the actual service providers may differ. Such a white-page service can utilize geographical information (or situation awareness [24]) and return locally tailored information. For example, the distance between a suitable agent and the user may be limited to a certain number of hops.

To summarize, a multicast agent can be a home agent, a "companion" agent (that travels together with a mobile host, e.g., an agent residing inside a laptop), or a foreign agent. A foreign agent can be located in one of the three ways:

- by home agent's mediation,
- through anycast, or
- via a White Pages service.

Note that a malicious server may send a bogus response to an anycast message, but will be discovered when the provided information is authenticated. Also, agents themselves can be mobile, and they can use point-to-point messages to keep in contact with their home representatives [40]. The collection of agents can in fact form a virtual M-bone – that is, a logical mobile network [31]. In this case, a mobile IP-multicast facility could simplify our design.

Finally, we note that detailed protocols for efficient and secure handing off to a new (e.g., neighboring) agent need to be developed. Moreover, combining a firewall

machine [8] with a secure multicast agent appears to be a natural solution that would allow multicast traffic to securely travel across firewalls.

5.4. Encryption and key management

A sender in a multicast session transmits a message to his secure multicast agent, which may choose to encrypt the traffic with a suitable session key before handing over the message to the (untrusted) multicast mechanism. At the receiving end, a client obtains information through subscription to his agent, just as one gets newspapers via a news agent. A session participant's agent examines all incoming multicast traffic to see if anything is of interest to the participant, and forwards information as is appropriate.

Often a correct decryption must be performed before the agent can decide whether to forward a multicast message or not. If encrypted traffic carries a session indicator, then the agent decrypts the traffic if it has the appropriate session key; otherwise, the agent must attempt to decrypt all incoming traffic to determine if any of it is legitimate – whether it can be decrypted properly and if it satisfies integrity check – and passes it on to a client. If the client is taking part in multiple sessions with different session keys, multiple decryptions may have to be attempted (in parallel or sequentially) before the agent can decide whether to pass along some traffic. Consequently, encryption algorithms must be sufficiently fast to handle the amount of traffic. Session indicators will be useful to traffic analysis, thus it should probably be up to individual sessions to decide whether to include them in the message headers.

A user or his machine, either directly or through an agent, joins a session and obtains the session key from the controller of that session, which is typically a multicast management center or a key distribution server. There are a number of ways to get in touch with the session controller. For example, the user can use the information provided in session advertisement. Or the user can query his favorite and trusted information server, which can very well be represented by or located via his home agent. The user can also look for the controller via anycast or the white pages service, just like locating an agent. Here, the authenticity of the reply must be verifiable – for example, via checking a signed and unexpired certificate.

Therefore, a mobile user can make all necessary preparations for taking part in secure multicast sessions through his home agent, as if he did not leave his home office. As a result, he need not carry a very complicated key-distribution package such as Kerberos [29]. A very simple protocol should be sufficient in most cases. In fact, if he plans to participate in a scheduled session, he can obtain a suitable key before leaving home and manually type it in on the road, much like using a sheet of one-time passwords for remote login.

5.5. Multicast with a group of mobile hosts

The agent-based architecture outlined above must be extended to accommodate a group of mobile hosts which share a common broadcast channel (e.g., wireless LAN) while participating in a multicast session. Examples for such a scenario include a meeting of executives who use their portable computers to complement their face to face negotiations. Clearly the architecture will not scale well if the group communication is conducted via individual unicast connections between each mobile host and its private agent. Efficient use of the wireless channel requires the broadcast of messages on the physical channel to all mobile users. This can be accomplished by the mobile users agreeing on a common agent and a group key and the broadcast of the information to the group using that key.

Agreement on a common agent by the mobile hosts is a part of the session setup procedure. Initially, each host begins with his own agent with whom he communicate by a unicast connection. However, whereas in the single-user case the migration of the agent functions is done to supports the goals of that user alone, in a group communication the agent functionality migrate to another agent who can function best for the group as a whole.

The objective of agent consolidation is to optimize the use of the wireless channel while maintaining an adequate security level for each of the participants. When the agent operates at the lowest trust level, i.e., no trust, the consolidation entails only the migration of the forwarding function from all the individual agents to a common point. This can be accomplished by one of the participants making the decision and the other following suit or by a distributed algorithms for reaching a consensus, e.g., leader election.

If the mobile hosts can decide on a common agent that they can all trust, a similar process of migrating to the common agent takes place. In this case, however, the process must involve transfer of the key management an encryption and decryption functions.

The most complicated scenario is when the mobile host require agent assistance in their security function yet they cannot agree on a single trusted agent for all their traffic. In this case a hybrid solution has to be developed, in which mobile hosts partition their traffic among the common trusted agent and their respective private agents.

6. Work related to multicast security

One of the earliest examples of work on process groups is [7]. In much of the later theoretical and practical work on broadcast protocols (e.g., [4]), the purpose is largely to achieve broadcast atomicity and/or to maintain causal or total ordering of messages, with no or little

consideration for security. Notably, the Isis/Horus group has recently incorporated some security mechanisms into their systems [35,36], and later extensions have been made to develop secure agreement protocols [33,34]. However, because the goal there is to maintain "virtual synchrony" when some group members can be corrupt, which is a much stronger goal than what we required here, these protocols are very complicated and suited only for small sized groups in terms of efficiency.

An alternative approach, in fact our approach here, is to implement these complex broadcast protocols based on a trusted multicast facility. In this way, multiple applications can share the same software platform, have consistent security assumptions and protocols, and can thus coexist. This approach also greatly simplifies the (repetitive) task of getting the security mechanisms right in all applications.

Another related work is IP-multicast [11,12]. Our trusted multicast facility can be based on IP-multicast and should not require modifications to IP-multicast or lower layers of the network architecture (except for preventing misuse of network resources). There are prior efforts addressing security issues in the Internet (e.g., [43,15,23,25,29]). However, there were no coherent architecture and protocols for secure multicast. The Visa protocol controls the movement of datagrams (and thus the use of network resources) on a per-user basis, whereas in trusted multicast the granularity may need to be finer.

Heterogeneous Multicast (HMC) [38] explores the concept of multicast sessions in which users participate at different bandwidth levels. This concept can be extended into multicast at different security levels.

7. Summary and future work

Advances in networking technologies have made multiparty communication an active area of research and development, where multicast is an important mode of communication as well as a good platform for building group-oriented services. We have considered fundamental security issues in building a trusted multicast facility, and we have discussed threats, new requirements for security, solutions, and trade-offs between scalability and security. In particular, we have outlined protocols for secure session registration, for session membership change, and for secure and efficient broadcast, as well as an architecture for secure multicast in mobile environment.

We are currently prototyping some of the techniques and protocols we have discussed in this paper. Such an experiment not only can bring valuable performance results, it can also establish a sound platform on which to build a wide range of more sophisticated protocols and trusted applications.

Acknowledgements

This paper is an extensive revision and extension of a previous paper presented at IEEE ICNP '94 [20] and also incorporates materials on secure broadcast from a paper presented at ACM CCS-2 [19].

References

[1] M. Abadi and R.M. Needham, Prudent engineering practice for cryptographic protocols, *Proc. IEEE Symp. on Research in Security and Privacy*, Oakland, California (1994) pp. 122–136.

[2] A.J. Ballardie, P. Francis and J. Crowcroft, Core based trees – An architecture for scalable inter-domain multicast routing, *Proc. ACM SIGCOMM*, San Francisco, California (1993) pp. 85–95. Published as ACM Comp. Commun. Rev. 23(4) (1993) 85–95.

[3] S. Berkovits, How to broadcast a secret, *Advances in Cryptology: Proc. Eurocrypt '91*, Vol. 547 of Lecture Notes in Computer Science (Springer, New York, April 1991) pp. 535–541.

[4] K.P. Birman and T.A. Joseph, Reliable communication in the presence of failures, ACM Trans. Comp. Syst. 5(1) (1987) 47–76.

[5] C. Blundo, A. De Santis, A. Herzberg, S. Kutten, U. Vaccaro and M. Yung, Perfectly-secure key distribution for dynamic conferences, *Advances in Cryptology: Proc. of Crypto '92*, Vol. 740 of Lecture Notes in Computer Science (Springer, New York, 1992) pp. 471–486.

[6] J.A. Bull, L. Gong and K.R. Sollins, Towards security in an open systems federation, *Proc. European Symp. on Research in Computer Security*, Vol. 648 of Lecture Notes in Computer Science (Springer, 1992) pp. 3–20.

[7] D.R. Cheriton and W. Zwaenepoel, Distributed process groups in the V kernel, ACM Trans. Comp. Syst. 3(2) (1985) 77–107.

[8] W.R. Cheswick and S.M. Bellovin, *Firewalls and Internet Security* (Addison-Wesley, 1994).

[9] G.H. Chiou and W.T. Chen, Secure broadcasting using secure lock, IEEE Trans. Software Eng. 15 (1989) 929–934.

[10] M. de Prycker, *Asynchronous Transfer Mode: Solution for Broadband ISDN* (Ellis Horwood, New York, 2nd ed., 1993).

[11] S. Deering, Host extensions for IP multicasting, Request for Comments 1112, Internet Network Working Group (1989).

[12] S.E. Deering and D.R. Cheriton, Multicast routing in datagram internetworks and extended LANs, ACM Trans. Comp. Syst. 8(2) (1990) 85–110.

[13] L. Delgrossi, R.G. Herrtwich, F.O. Hoffman and S. Schaller, Receiver-initiated communication with ST-II, Preliminary version (1994).

[14] W. Diffie and M.E. Hellman, Exhaustive cryptanalysis of the NBS data encryption standard, IEEE Comp. 10(6) (1977) 74–84.

[15] D. Estrin, J.C. Mogul and G. Tsudik, Visa protocols for controlling interorganizational datagram flow, IEEE J. Select. Areas Commun. 7 (1989) 486–498.

[16] H.T. Kung et al., Secure short-cut routing for mobile IP, *Proc. USENIX Summer Technical Conference*, Boston, Massachusetts (1994).

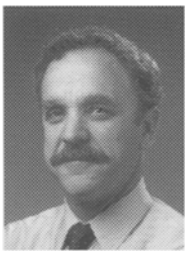
[17] The ATM Forum, *User-Network Interface Specification (Version 3.0)* (Prentice-Hall, New Jersey, 1993).

[18] L. Gong, Efficient network authentication protocols: Lower bounds and optimal implementations, Technical Report SRI-CSL-94-15, Computer Science Laboratory, SRI International, Menlo Park, California (1994).

[19] L. Gong, New protocols for third-party-based authentication and secure broadcast, *Proc. 2nd ACM Conf. on Computer and Communications Security*, Fairfax, Virginia (1994) pp. 176–183.

[20] L. Gong and N. Shacham, Elements of trusted multicasting, *Proc. IEEE Int. Conf. on Network Protocols*, Boston, Massachusetts (1994) pp. 23–30. A preliminary version appeared as Technical Report SRI-CSL-94-03, Computer Science Laboratory, SRI International, Menlo Park, California (1994).
[21] L. Gong and D.J. Wheeler, A matrix key distribution scheme, J. Cryptology 2(2) (1990) 51–59.
[22] G.H. Hardy and E.M. Wright, *An Introduction to the Theory of Numbers* (Oxford University Press, Oxford, England, 1979; first ed. 1938, fifth ed. 1979, reprinted (with corrections) 1983).
[23] C. Huitema and B. Braden, Report of IAB workshop on security in the internet architecture, Internet draft, IAB (1994).
[24] R.H. Katz, Adaptation and mobility in wireless information systems, IEEE Personal Commun. (First Quarter 1994) 6–17.
[25] S.T. Kent, Internet privacy enhanced mail, Commun. ACM 36(8) (1993) 48–59.
[26] C.S. Laih, J.Y. Lee and L. Harn, A new threshold scheme and its application in designing the conference key distribution cryptosystem, Inf. Proc. Lett. 32 (1989) 95–99.
[27] J. McLean, The specification and modeling of computer security, IEEE Comp. 23(1) (1990) 9–16.
[28] M. Minsky, A conversation with Marvin Minsky about agents, Commun. ACM 37(7) (1994) 23–29. Interviewed by Doug Riecken.
[29] B.C. Neuman and T. Ts'o, Kerberos: An authentication service for computer networks, IEEE Commun. 32(9) (1994) 33–38.
[30] C. Partridge, T. Mendez and W. Milliken, Host anycasting service, Request for Comments 1546, Internet Network Working Group (1993).
[31] C.E. Perkins and P. Bhagwat, A mobile networking system based on internet protocol, IEEE Personal Commun. (First Quarter 1994) 32–41.
[32] B. Preneel, Cryptographic hash functions, European Trans. Telecom. 5 (1994) 431–448.
[33] M. Reiter, A secure group membership protocol, *Proc. IEEE Symp. on Research in Security and Privacy*, Oakland, California (1994) pp. 176–189.
[34] M. Reiter, Secure agreement protocols: Reliable and atomic group multicast in rampart, *Proc. 2nd ACM Conf. on Computer and Communications Security*, Fairfax, Virginia (1994) pp. 68–80.
[35] M. Reiter, K. Birman and L. Gong, Integrating security in a group-oriented distributed system, *Proc. IEEE Symp. on Research in Security and Privacy*, Oakland, California (1992) pp. 18–32. Also available as TR92-1269, Department of Computer Science, Cornell University.
[36] M. Reiter and L. Gong, Preventing denial and forgery of causal relationships in distributed systems, *Proc. IEEE Symp. on Research in Security and Privacy*, Oakland, California (1993) pp. 30–40.
[37] R.L. Rivest, A. Shamir and L. Adleman, A method for obtaining digital signatures and public-key cryptosystems, Commun. ACM 21(2) (1978) 120–126.
[38] N. Shacham, Multicast routing of hierarchical data, *Proc. Int. Conf. on Communications*, Chicago, Illinois (1992).
[39] A. Shamir, How to share a secret, Commun. ACM 22 (1979) 612–613.
[40] W.A. Simpson, IP mobility support, Draft 4, IETF Network Working Group (1994).
[41] C. Topolocic, Experimental internet stream protocol, Version 2 (ST-II), Request for Comments 1190, Internet Activities Board (1990).
[42] Data Encryption Standard, (U.S.) National Bureau of Standards, (U.S.) Federal Information Processing Standards Publication, FIPS PUB 46 (1977).
[43] V.L. Voydock and S.T. Kent, Security mechanisms in high-level network protocols, ACM Comp. Surveys 15 (1983) 135–171.

Li Gong received the B.S. degree with honors (1985) and the M.S. degree (1987) from Tsinghua University, Beijing, China, and the Ph.D. degree from the University of Cambridge (Jesus College, 1990), England, all in computer science. His research is in distributed systems and communication networks, particularly in issues of fault tolerance and security. He was Program Chair of the 7th and 8th IEEE Computer Security Foundations Workshops (1994 and 1995), is Program Co-Chair of the Third ACM Conference on Computer and Communications Security (1996), and is on the editorial board of the Journal of Computer Security. He received the IEEE Communications Society Leonard G. Abraham Prize Paper Award in 1994 and the IEEE Computer Society Symposium on Security and Privacy Outstanding Paper Award in 1989.



Nachum Shacham received his B.Sc. and M.Sc. degrees in electrical engineering from Technion, Israel Institute of Technology in 1970 and 1976 respectively, and his Ph.D. degree in electrical engineering from the University of California at Berkeley in 1980. He is presently Program Director for networking research at the Computer Science Laboratory, SRI International, where he has worked on such areas as packet radio networks, adaptive error-control codings, heterogeneous multicast (HMC), and adaptive network algorithms. He is a Fellow of the IEEE, is on the editorial board of a number of technical journals, and was Program Chair of IEEE INFOCOM '91.