SMIF: A Framework for Secure Multicast Intercommunication

Dawn Song Yang-hua Chu Adrian Perrig

De,ember 20, 199,

Contents

1 Introduction

In the view of the constantly expanding Internet, more applications make use of the Internet every day. Internet telephony, video conferencing, up-to-date stock quotes, etc. just to mention a few. Extending an application to make use of the Internet may greatly increase the functionality but on the other hand security and privacy issues arise. Anybody on the planet might interfere or cause problems.

Since security is a difficult problem to solve, new Internet applications or protocols postpone security issues until later versions. The same observation applies to multicast - the first implementations did not address security. We believe that security is important and a necessary component for many serious applications. Therefore we feel that adding security and privacy capabilities to multicast of high importance.

In this report we describe the SMIF framework, an environment that supplies a system designer with building blocks that let him realize a wide variety of different security requirements. In addition the designer also has a simpler task as he does not need to be familiar with security subtleties. This approach leads to a more complicated environment than a simple off-the-shelf solution. Because there is no easy method that solves all combinations of security requirements optimally our approach is the right way to go.

Terminology

To illustrate the protocols we will use our friends Alice and Bob to describe the good users and Trent acts as a trusted server. Unfortunately in the real world we also have evil and malicious personalities, represented by Mallory who plays the malicious attacker and Eve who likes to eavesdrop. These personalities are also described in [Sch96].

The terminology used to describe multicast participants is *senders* or *receivers/subscribers* if we want to distinguish between the capability of sending vs receiving and we just use *member* if the distinction is not important in the context. A user may *join* or *subscribe* to a multicast group. He can either *leave* the group voluntarily or get *expelled.*

The notation for cryptographic methods are the following. When we encrypt a message M with key K we write {M}K. The hash of message M is denoted as H(M).

2 Motivation

In today's Internet many applications demand security related features. In the view of new Internet services, such as video distribution or video conferencing, multicast will become an important technique to reduce network traffic. Because of the possibly large number of subscribers and the possibility for anybody to join the group, security becomes crucial for the correct behavior of multicast applications. Let's take a look at various applications and their security requirements.

- For a military secure channel we would like to have the highest secrecy (confidentiality) and authentication achievable. It also requires protection from traffic analysis.
- Commercial organizations require confidentiality and authentication for their secure communications such as video conferencing.
- Some organized chatting groups such as a virtual classroom need sender authorization to protect from malicious spamming of the group.
- An unorganized group such as non-arbitrated chatting-room may want both sender and receiver anonymity and group confidentiality.
- Commercial data/information distribution such as stock market data, weather broadcast or other database querying need sender authentication and atomicity with billing. Some receivers may want to have privacy (anonymity) too.
- Internet casinos may require customer anonymity and fairness for the game.

3 SMIF Requirements

In section 2 we have seen that applications demand various levels of security as well as different security properties. We have established a list of security capabilities of SMIF that are fully supported. Sections 5 to 7 explain how to use the SMIF basic building blocks to achieve any combination of security requirements.

The security requirements we address in SMIF are: confidentiality, privacy of sender and receiver/subscriber, source integrity, authentication of the sender, non-repudiation and sender authorization.

Confidentiality

Confidentiality is also known as *destination security* or *secrecy of data.* Eve the eavesdropper should not be able to infer any information about the content by reading data packets on the network. Only the person for which the data is destined is able to infer meaningful information from data packets.

More specifically, a multicast message that implements confidentiality must therefore only consist of encrypted information. All subscribed users share a common secret that allows them to decrypt the information.

[Mit97] addressed the issue that new group members should also not be able to decrypt previous information of the multicast group. The shared secret must therefore change each time a new member joins the group.

Similarly a subscriber that leaves the group should not be capable of decrypting subsequent messages. Especially when the member is expelled by the group. Clearly, the shared secret must also change in this case.

Anonymity of Sender and Privacy of Receiver

In the way the Internet is used predominantly today, many users wish to retain their privacy. We can distinguish between privacy of the user with respect to other users on the same local network or Internet for receiving information and anonymity for sending information.

Unfortunately privacy and anonymity are not end-to-end relations such as confidentiality. When we encrypt a message to receive confidentiality the lower network layers don't need to provide encryption. But to reach anonymity **all** of the layers need to provide it. If only one layer violates anonymity we loose it for all layers. Since SMIF is mostly independent of the underlying layers we cannot guarantee absolute anonymity. Instead we will rely on a trusted third party to achieve anonymity. To attain privacy we will encrypt the data so no eavesdropper can infer what information the subscriber is looking at. But again, the underlying layers may violate the privacy. For example an eavesdropper may find out the multicast group a user is subscribed to through the IGMP protocol. He can then subscribe to the group as well to "eavesdrop" directly.

Another privacy issue is linked to expelling of members. In the case where multiple members are expelled at once, the expelling step should be atomic and should not allow any member in the group to find out the identities of the other expelled members. We would like to prevent scenarios where expelled members might collaborate to exchange information to stay in the group. Another issue is where a group member directly contacts an expelled member to offer some "special deal".

Source Integrity

Data integrity is an important property required by many applications. Sensitive data that would give an attacker an advantage by changing it, must therefore be protected. By requiring a multicast message to satisfy source integrity we express that nobody can change the information between the source and the destination.

This can be especially useful for news distribution, stock market quotes or even advertisements in articles we read. In the last case mentioned an ISP could potentially make profit by replacing commercial Web pages and changing the advertisements as the pages are forwarded to the client. The ISP could make profit by charging customers for insertion of their advertisements.

Authentication of Sender

The past has shown that impersonation or obfuscation of the identity is easy to achieve in TCP/IP systems. Especially the IP address spoofing or TCP SYN flooding attacks both rely on wrong IP source addresses. Multicast groups carrying sensitive information need to protect the senders from impersonation and the receivers from falsified data.

The classical solution to these problems is to add an authentication mechanism to the protocol. This allows the receiver to unambiguously authenticate the sender of every message, preventing counterfeiting.

Non-repudiation of message

In today's Internet, message packets are routed through many untrusted domains. A malicious attacker sitting somewhere in the network might change the contents of information. This could be fatal for sensitive data such as stock market quotes or medical documents.

The receiver therefore wants to make sure that the data was really sent by the sender and has not been changed underway. Integrity of the message in conjunction with authentication of the sender achieve these requirements.

In another setting the sender wants to have a proof that a certain message was really sent by a certain person. We call this non-repudiation of the message. This can be useful for example for electronic commerce protocols or web server replies where the client demands legally binding responses[1].

We would like to add that both of the above settings are equivalent. In either case we need the non-repudiation property of the sent messages. This can be achieved by using the integrity of messages as well as authentication of senders described above.

Authorization of Sender

Another requirement of some multicast groups is the authorization of the sender. Any subscriber only accepts messages from authorized senders. One of our design rationales was not to modify the underlying multicast protocol. We have therefore the problem that any member of the multicast group can send messages to the group and every other member will receive them. Since we are not changing the underlying multicast protocol in SMIF, we have no way of preventing spammers (unauthorized senders) directly. But we provide for a mechanism that makes it easy for any receiver to verify the authorization of a sender to send a message.

Scalability

Scalability becomes an issue in any distributed system which involves a possibly large number of users. Especially in multicast we might have millions of subscribers in one multicast group if we consider Internet television. Under this viewpoint we need to be very careful to design the algorithms accordingly. For example if our key distribution scheme is quadratic in the number of group members n, we would have a poor scalability for large groups.

Our design rationale for SMIF therefore involves good scalability to make the system work even if the group is highly dynamic and millions of subscribers join. We want to keep the computation overhead low as well as the state storage overhead necessary for the security.

Robustness

Since no network is perfect SMIF must take into account that any router or host may fail. The security must not rely on assumptions on the network reliability. Conversely we also want to retain functionality even if the network has a non-negligible error rate.

[1]We assume in this case that the web server sends its most frequently read pages over a multicast group

Basically speaking, SMIF should remain functional even if one or many main components fail. We can achieve this by adding redundancy. A single point of failure is therefore unacceptable.

Multicast Protocol Independent

SMIF is a high-level design and does not rely on specific underlying protocol features. Further we don't change the underlying multicast semantics.

4 SMIF building blocks

In section 3 we have established the necessary requirements for secure multicast. In this section we present a set of building blocks that we use to realize any combination of security requirements. The blocks we use are reliable multicast, cryptographic algorithms, group key management and a trusted server.

Cryptographic Algorithms

The problem of achieving confidentiality is solved by using encryption techniques as described in [Sch96]. For speed issues we will use symmetric algorithms such as IDEA or DES. In the case where we need asymmetric encryption we will still use a symmetric algorithm to encrypt the message with a new random key and we only encrypt the random key with the asymmetric method, resulting in a large speedup.

The asymmetric algorithm we use is RSA with a minimum of 512 bits or an elliptic curve algorithm. RSA allows us to encrypt information as well as producing digital signatures. The RSA public keys are distributed by using an existing public key infrastructure as described in the following paragraph.

A public key infrastructure aims to bind a principals identity to a public key. A principal, which can be a human or an organization, register its identity and the corresponding public key to some trusted entities in the infrastructure. The trusted entities, who set and publish guidelines, certify new principals who comply with the guideline, and validate the binding upon request. This is the basic assumption by well-known public key infrastructures such as X.509 and PGP. The difference is in turns of architecture; namely how these certification authorities interact with each other to make it scalable to a large number of principals.

Currently there are two widely used public infrastructure, X.509 and PGP. X.509 has a hierarchical structure equivalent to a tree. The root is called the Internet Policy Registration Authority (IPRA). Beneath the IPRA are Policy Certification Authorities (PCA), each of which establishes and publishes its policies for registration of users or organizations. PCAs in turn certify CAs, which in turn certify subordinate CAs, users, or organizations. When user A wants to authenticate user B's public key, user A finds the proper certification path by traversing up the certification hierarchy until a mutual CA is reached and then traversing down the hierarchy until user B is reached.

PGP adopts the "Web of Trust" model, where each principal can also a CA, and a network of principals forms a public key infrastructure. To establish a certification path from user A to B, user A queries a list of trusted principals about user B, which can in turn query their trusted principals, until user B is reached. The problem of PGP is the assumption that trust is transitive. If user A trust user B's public key is correct, it does not imply user A trust user B's judgement on user B's list of trusted public keys.

To achieve the message integrity we need a hashing algorithm. Since MD5 has been proven not to be robust enough against collisions we will use SHA or RIPE-MD.

Server based

To maintain a secure multicast group, we need to address several issues. The first one is how to establish the group. The second one is how a person joins the group, ie. the policy for joining. The third one is that for people who are already in the group, who can send the messages and who can listen to which kind of messages. The forth one is that what need to be done when some people leave the group. We choose a centralized scheme to address all these issues. For the broadcast scheme where only one sender can send such as magazine distribution where the publisher or the sender will be the server to start up the group and decide who can join the group and who need to leave the group and all that kind of policies. For the multicast scheme, we have an explicit server to establish and enforce these polices.

Key management

The next building block is the key management. A person needs a shared secret or a public key certificate to authenticate itself to the server to join the group. It also needs a group session key to decrypt the messages. In a sender authorization scheme, senders also need a sending key. In a sender authentication scheme, senders need keys to sign the messages. To achieve a small cost for updating group session keys, we need some keys to encrypt the new group session key. The requirement of the signature and encryption functionalities make it difficult to design a scalable and secure key management scheme.

One solution for authentication of sender (or client) is to use the public key infrastructure in which everybody has its own public/private key pair. When a user wants to join the multicast group, it will use its private key to authenticate to the server to get the group key like the X.509 or other public key structures. The server will assign keys to the client based on the authentication of the client and its access control list. Also the authentication of senders in this structure seems very straightforward. A sender can simply sign its message with its private key and multicast it. The disadvantage for this is that the public key structure is usually very expensive. We need to trust a third party, namely the Certification Authority (CA) to authenticate the users and create valid certificates. Public key encryption and decryption are also an order of magnitude slower than the equivalent private key algorithms. But the advantage is that by having a public/private key pair, a client basically can authenticate to anybody else by the public key certificates.

The second solution for authentication of sender (or client) is the private key infrastructure. For example, after the client pays the money, the server will establish a symmetric key between itself and the client. Later the client will use its symmetric key to authenticate itself to the server. In general, symmetric key cryptography has a low overhead and keys are easy and very fast to generate. But a symmetric key structure can only let the client to authenticate itself to the server since it's a shared key between the itself and the server. So for sender authentication, we can either use the public key as we described in last paragraph, or use the private key structure in which the sender will send the message signed by its symmetric key to the server first, and then the server will multicast the message signed by its own private key including the sender's name.

To do group session key updates, one way is that the server encrypts the new group session key with every trusted member's public key or symmetric key and then unicast it to the trusted member. But in this case the server needs to send N messages for one key update which is obviously not scalable. To decrease the cost, we use the approach presented in [WHA97]. The key server keeps a key hierarchy of symmetric keys as shown in figure 1. Each user at the leaf node of the tree knows all the keys on the path to the root.

For example user 2 knows key E, B and A. Key A is used for data encryption in the multicast group. In case that a key needs to be changed, the new key only needs to be encrypted with the two children keys and re-sent.

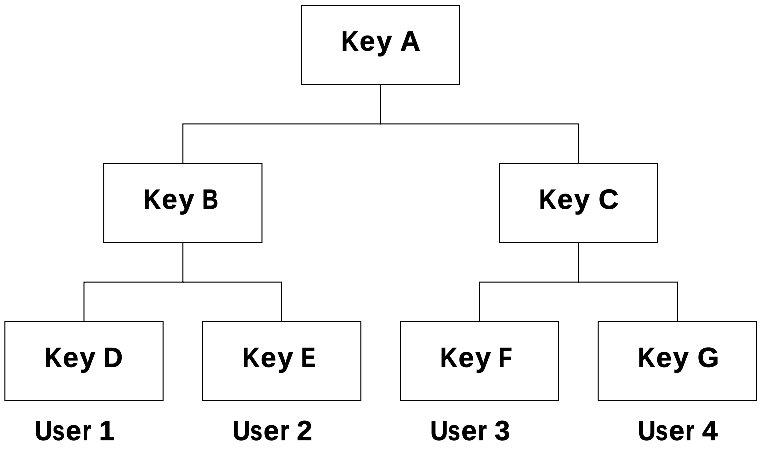

Figure 1: Key management scheme

This scheme makes group re-key very cheap. For example when a new user joins, he gets all the keys from his leaf node up to the root. It is easy to see that the number of keys each user needs is logarithmic in the number of members. This is because the height of the tree $h = \log_2(n)$ with n as the number of group members. In case a member leaves the group, we need to change all the keys the member has. The re-key is made by changing all keys from the leaf node up to the root. Let's explain how this works with an example. Let's assume user 3 leaves the group. This implies that we need to change keys C and A. Since key F is only used for user 3 we don't need to change it. To change key C, the server encrypts the new key C' with key G. To change key A, the new key A' is encrypted with the old key B and the new key C'. Therefore every member except user 3 has again all the necessary group keys.

This scheme scales well for even a large number of members. We can see that even if we have 10^9 members, each member only needs 30 keys. Each join or leave also needs only 60 encryptions. Unfortunately the number of keys that need to be managed by the server is $2 * n$. The number of key changes is also substantial for highly dynamic groups. But we assume that generating good fresh random symmetric keys is feasible.

Reliable Multicast

Because there are security-critical information (such as group secret key) being exchanged over the multicast protocol, we are building our security multicast framework on top of a reliable multicast service. It is worth noting that the multicast messages related to maintaining secure service may piggy back on top of the regular multicast messages. Since most reliable multicast services are initiated by receiver, receiver has a choice not to recover the lost message if the receiver determines that the multicast message is not critical.

Providing a reliable multicast is still under heavy research. Proposed schemes include SRM described in [FJM+95] and STROM [XMZY97].

Alternatively, we may choose to replace the assumption of a reliable multicast service with unreliable multicast service, and have a reliable unicast as a channel to convey security-critical information that is lost in the multicast protocol. However, we speculate that when the multicast group grows large in size, the chance of missing packets increase up to a point that it becomes more economical to use existing reliable multicast than opening individual reliable unicast sessions.

5 Simple Architecture

In this simple architecture, we are exploring a secure multicast architecture that supports integrity and authenticity properties in an efficient manner.

5.1 Scenario

Advertisement revenues is recognized as the major financial sponsorship of the free (and valuable) information on the Internet today. To ensure that their money is paid for, advertisement agencies have strong demand that the advertisements be delivered to the receiver without error or modification. Potentially, malicious Mallory could swap the advertisements during transmission and make money with the injected advertisements.

5.2 Architecture and Protocol

This architecture requires a reliable multicast to deliver both the multicast messages and their associated security information. The picture looks like the following:

Alice, the sender, has a public/private key pair for the digital signature. Alice publishes her public key using the public key infrastructure. Bob, the receiver, can verify Alice's public key using the same public key infrastructure. Note that Bob is anonymous because he does not need to authenticate to Alice to receive the broadcast message.

The protocol runs as follows:

1. For a given message M, a given one-way hash algorithm H, and a given signature key pair, (K.Alice.priv, K.Alice.pub), Alice signs the hash of the message, and broadcasts to the group:

   $$\text{Alice} \Rightarrow \text{Group: [M, H(M), \{H(M)\}K.Alice.priv]}$$

2. Bob, upon receiving the message, verifies that the signature is correct based on his knowledge of Alice's public key, and that the hash H(M) corresponds to the message M.

5.3 Analysis

As a malicious middle man, Mallory cannot alter H(M) because it is protected by Alice's signature. Therefore, the integrity of M is preserved by the hash function H. Moreover, Mallory cannot pretend to be Alice because he cannot construct Alice's signature.

Digital signature works well if the broadcast message is small and is completely known at the time of broadcasting (so Alice can compute the hash). However, messages such as video streams or live stock quotes do not fit well with conventional digital signature. As a replacement, we can use the stream digital signature as introduced in [GR97].

This architecture is described as a broadcast application. However, we can easily translate the architecture into a multicast application, where a member plays both roles as Alice (the sender) and Bob (the receiver). The main problem is when the multicast group gets very large, each receiver must keep track of all the public keys necessary to authenticate all the senders in a multicast group. Moreover, this architecture fails to address some other important properties such as sender authorization and confidentiality.

6 Architecture With Confidentiality

We extend the previous architecture to include confidentiality. Under this architecture, every member in the multicast group can send and receive messages.

6.1 Scenario

Secure closed-door video conferencing will be an important application to conduct business meetings over the Internet. The "close-door" policy implies a strong confidentiality requirement. No one other than the group members may send messages to and receive messages from the group. We therefore get a weak authorization since only group members can send valid messages to the group. The authorization is weak because we don't have a fine granularity for sender access control.

By encrypting all group communications we get anonymity and privacy to eavesdropper. Only if any eavesdropper has the possibility to join the group we loose anonymity. The anonymity is therefore dependent on the group join policy.

6.2 Architecture and Protocol

This architecture uses reliable multicast and the group key management protocol described in section 4 as the main building block. Under group key management protocol, Trent is the trusted server who is responsible for admitting and expelling members, while updating fresh group keys during the process. With a fresh group key (K.Group), the protocol is quite simple:

1. Alice creates the hash of the message H(M) and encrypts together with her message using K.Group. If authentication is necessary (and hence her identity is revealed), Alice puts her signature of H(M) before the encryption. Then she multicasts the encrypted message to the group:

   $$\text{Alice} \Rightarrow \text{Group: } \{\text{M, (H(M) or } \{\text{H(M)}\}\text{K.Alice.priv)}\}\text{K.Group}$$

2. Bob, a member of the group, decrypts the message, verifies that the hash matches the message and that Alice's signature is correct (if available).

6.3 Security analysis

It is easy to see that integrity is guaranteed by the hash function and the authenticity is guaranteed by Alice's signature. If Alice does not provide her signature, the authenticity is a weaker statement: "this message comes from a group member who holds the group key".

Confidentiality is also preserved under this protocol. As long as a valid group key is established, any member can encrypt the message with the group key before multicasting it to the group. Any other members in the group can decrypt the message with the same group key, but no eavesdropper can.

At our secure multicast protocol level, Alice's anonymity is preserved inside the group if she does not sign with her signature. Her anonymity is preserved outside the group regardless because no eavesdropper can decrypt the message. Since anonymity is not end-to-end as mentioned in section 3 we can't guarantee perfect anonymity and privacy because of traffic analysis on lower layer protocols.

6.4 Performance analysis

In section 4 we have shown that the group key management is scalable because it requires only logarithmic overhead for each join and leave. All the new keys can be encrypted and concatenated in one message which is sent to the multicast group. The new member gets a short unicast message from the server. If we have 10^9 members we have seen that we only need 30 keys. We know that symmetric keys are very short, the longest ones today use 20 bytes. Symmetric encryption does not make the message longer than one block-size, which is usually 8 bytes. Therefore the message to the new user will be at most 24 bytes * 30 which is only 720 bytes. The message which is multicasted carries each new key encrypted twice. Therefore this message is only 1440 bytes long. Considering that these calculations are for one billion subscribers, we can see that the key management overhead on the network is negligible.

Further we can argue that we can also cluster joins and leaves. This will make the overhead much smaller when we cluster simultaneous joins in one subtree since only one update message needs to be sent to the multicast group. In case leaving members are also localized in the tree, the key update can also get combined and the update only needs to start from the tree node that the leaving members have in common. The requirement for *atomic* expellation of subsets of members is therefore also satisfied since no expelled member stays "longer" in the group than any other.

We have shown that the performance overhead for key management functions is small. Next we investigate how much overhead the encryption and decryption yields. It is widely known that symmetric encryption algorithms are very fast to compute. On a Pentium-II based machine we can encrypt and decrypt at 10 Mbit/s in software. Since the ciphertext is not longer than the plaintext, security adds no network overhead[2]. Because a digital

[2]Only if we use compression algorithms the performance will be much lower with en-

signature is also very short (on the order of 20 bytes) this also does not add any considerable or unscalable overhead either.

7 Architecture 3

In this extended secure multicast architecture, we are trying to address all the desired security properties listed in section 3, namely integrity, authenticity, sender authorization, sender anonymity, receiver privacy, and most importantly, confidentiality.

7.1 Scenario

In Internet gambling, we want anonymity of players. Players don't want other people to know that they are playing or they win a million dollars. For most of the games, we also want confidentiality since fairness is dependent on this. We also want the authentication of the dealer.

For the military secure communication, usually secrecy has different levels. For different levels of secrecy, different levels of receivers can decrypt the messages. We also want non-repudiation of messages, authentication and authorization of senders. Since it's for military security, we don't want any privacy or anonymity of senders.

For the electronic magazines, we want privacy of the subscribers. Also we want the authentication of senders (which are the publishers). We also want the confidentiality of the data so nobody can read it without paying for it. And we want integrity so nobody can change the data on its way.

7.2 Architecture

This architecture uses reliable multicast and our group key management protocol as the main building block. In the group key management protocol Trent acts as the trusted server and is responsible for admitting and expelling members, while updating group keys during the process. In addition, Trent maintains an access control list of authorized senders. Upon receiving a "send" request, Trent broadcasts the message to the group only if the sender is in the access control list. No one other than Trent can broadcast to the group.

cryption. The solution here is to compress the message before encryption.

7.3 Protocol

At the beginning of the protocol, we assume the following:

- All members in the group hold the same symmetric group key K.Group
- Alice holds a private key K.Alice, shared only with Trent.
- Trent holds a public and private key pair (K.Trent.pub, K.Trent.priv). K.Trent.pub is known by all members in the group.
- Trent holds a list of authorized senders.

To send a message to the group, the following steps are performed:

1. Alice sends an encrypted unicast message to Trent using her private key K.Alice

   Alice -¿ Trent: {M}K.Alice

2. Trent decrypts the message, verifies that Alice is one of the authorized senders, encrypts the message with the group key, signs the encrypted message with his private key, and broadcasts to the group:

   Trent -¿ Group: [{M}K.Group, { H({M}K.Group) }K.Trent.priv]

3. Bob, one of the members in the group, verifies that Trent's signature is correct, and decrypts the message using the group key.

7.4 Security analysis

It is quite easy to see that integrity, sender authorization, and confidentiality properties are preserved under the protocol. Integrity is implicit with the use of symmetric key cryptosystem as long as there are enough redundancies in the message in which the receiver can verify. Authorization of sender is guaranteed by Trent's access control list. Confidentiality is guaranteed by the Alice's secret key in step 1 and group key in step 2.

It is less obvious what sender authenticity means in this protocol. In step 1, Alice's message is authenticated by the secret key K.Alice shared between Alice and Trent. In step 2, the Trent's signature simply means "the sender of this message is authenticated by me and is authorized to send you this message". To provide conventional message authenticity, Alice can sign her message at step 1 and later Trent broadcasts the signature with the message.

Alternatively, Trent can reveal Alice's identity with the protection of his digital signature in step 2. Note that there is a conflicting interests between authenticity and anonymity; our protocol preserves sender anonymity while maintaining some level of authenticity.

At the protocol level, Alice's anonymity is preserved as long as Trent does not explicitly leak out Alice's identity. Unfortunately there is a catch. Unlike encryption, anonymity is not end-to-end; Alice's identity may be leaked at every level of the protocol stack, such as IP unicast to Trent. The same argument holds true for receiver privacy. Although this protocol does not leak any privacy information about Bob, a potential eavesdropper Eve may listen to Bob's LAN traffic and learn that Bob belongs to the multicast group at the lower protocol layer (such as IP multicast and reliable multicast). We will discuss this further in section ???.

7.5 Performance Analysis

In architecture 2 construction we have shown that group key management protocol is scalable with logarithmic overhead for each join and leave (section **??**). Moreoever, the

7.6 Alternative Architecture

The previous architecture makes Trent, the server, a hot spot, since every message must be decrypted by Trent to verify authorization and reencrypted it again to preserve confidentiality. This architecture allows a sender to send multicast messages directly to the group.

To satisfy sender authorization requirement, we made the following simple hack:

We strategically place all authorized senders under one subtree (the subtree under Key B) which is separate from the rest of the members who are authorized to receive messages only. The sender's key (in this case Key B) is assymetric (K.Sender.priv, K.Sender.pub). K.sender.priv, known by authorized senders only (User 1 and 2), is used for signing the message. K.sender.pub, known by all members in the group, is used for verifying that the message is sent by authorized senders.

This assymetric sender's key is updated everytime an authorized sender joins or leaves the multicast group. The server also needs to broadcast the new sender's public key (K.Sender.pub) to the rest of the group by encrypting it with the group key ({K.Sender.pub}K.Group).

Now the rest of the protocol becomes simple:

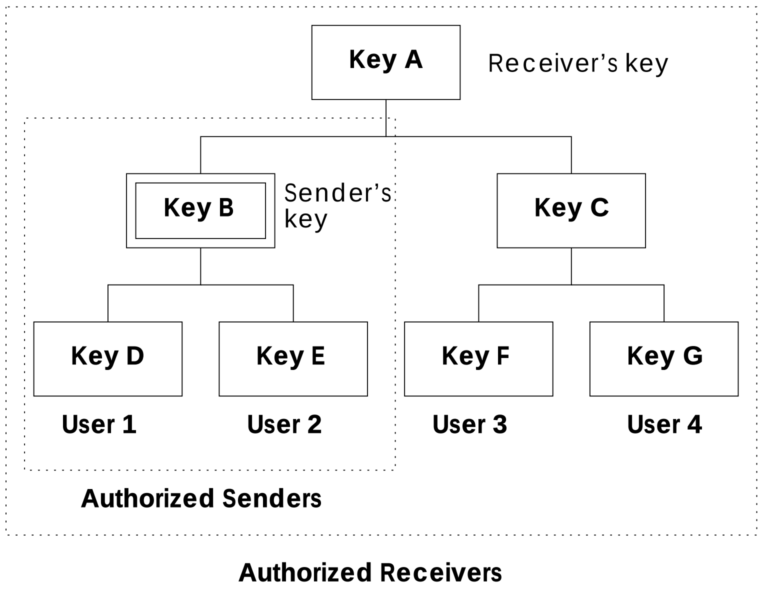

Figure 2: Key management protocol with sender's key

1. Alice encrypts the message using K.Group, signs the hash of the message using K.Sender.priv, and multicasts them to group:

   Alice -¿ Group: [{M}K.Group, {H(M)}K.Sender.priv]

2. Bob, a member of the group, decrypts the message, verifies the signature is good using K.Sender.pub, and the message matches the hash.

All the properties in the original architecture are preserved here. Integrity is guarenteed by the hash, which is in turn protected by the sender's private key. Sender authorization is achieved because only authorized sender can produce the signature. Confidentiality is preserved because it is encrypted with the group key.

Authenticity retains the similar meaning as the original architecture, namely "the sender of the message was authenticated by the key management server as one of the authorized senders of this multicast group".

Sender anonymity is stronger in this architecture, because the sender Alice does not need to expose her own identity to Trent to send a message. She can just sign the message and the rest of the multicast group will accept her message. Receiver privacy remains unchanged in this case.

8 Extension based on Active Networks and IPv6

Nowadays, the Internet is vulnerable to denial of service attacks such as flooding or spamming. And the development of multicast makes this attack even simpler and more powerful. A malicious host can just send any junk to the multicast group address. It can cause a big traffic load and all the end-hosts in the multicast group will be annoyed. One possible solution is that we can use sender authorization. So when the packet is forwarded from the network layer to the application layer, the application layer will do the authorization of the sender. If it's an unauthorized sender, the application will automatically drop the packet. In this case, the end-user won't notice the garbage data, but the computation power of the end-host will be wasted as well as the network resource. Also the billing based on network usage will be difficult since client won't like to pay for the garbage. So it will be more and more urgent to find a solution to prevent spamming.

Since we can't prevent malicious hosts from sending garbage packets, the spamming problem can't be solved end-to-end. The only way to prevent this is that the Internet routers can drop unauthorized packets. The current routing process forwards a multicast packet based on its routing table for this multicast group. This is called passive forwarding. Instead, we need intelligent routers to be able to check the validity of the packets. Via research on IPv6 and active networks, we propose a solution using the IP authentication header and the active network.

IPv6 includes security in its design goal. One new mechanism is the IP Authentication Header [Atk95]. End-hosts can either use a symmetric or asymmetric authentication algorithm to sign the entire IP packet except for the fields or the options that will be changed in transition (e.g. "hop count"). A separate IP Authentication key management component is used to create and maintain a logical table containing the security parameters for each current security association. Upon reading the security parameters from the logical table, a host can do the authentication on the datagram containing an Authentication Header. The receiver can use the Authentication Header to authenticate the sender of the packet.

In active networks [WGT98], intermediate routers are intelligent. Instead of doing passive forwarding, routers can do computation and processing according to the corresponding packet protocol. Though the security concern and the processing cost of the packet throw doubts on the development of active networks, the rich functionality which can be provided is very attractive. In our approach, the routers can do the IP Header Authentication and drop the unauthenticated packets.

Using IPv6 and active networks, we propose a solution for preventing spamming in secure multicast. The solution differs a little bit from the architecture we gave earlier in this paper.

For architecture 1, since everybody can send and the receiver will decide who it wants to listen to by its own access control list, there's no need to prevent spamming inside the network.

For architecture 2, all the group members share the same group session key and everybody in the group can send. We want to prevent from malicious people who are outside of the group to flood the multicast group. So in addition to the group session key, the server also generates a group authentication key which is used in generating the IP Authentication Header and distribute it to every group member when it joins the group. And several changes need to be done for the protocol. When a group member wants to send a message, after the normal encryption using the group session key, the IP layer needs to add the IP Authentication Header using the group authentication key. On the receiver side, the IP layer will first check the IP Authentication Header before it hands the packet to upper layers. The server will also need to register an entry in the IP Authentication Key Management's logical table that this group authentication key is associated with this multicast group address. When the group key is updated, the group authentication key also needs to be updated. The server will send out the new group authentication key together with the new group session key using the same method as described before which almost requires no extra cost. But the server also needs to update the entry in the IP Authentication Key Management's logical table. The routers will do IP header authentication based on the group authentication key. So if people outside of the group send packets using this multicast group address as destination address and didn't do the IP authentication, the router will drop the packet when it sees it. Therefore, we can prevent spamming to this multicast group.

For architecture 3, only authorized people in the multicast group are allowed to send. We gave two schemes in architecture 3.

One scheme is all the authorized senders share the same sending key (public key). To prevent from spamming, the server also needs to generate a group sender authentication key which is used in the IP authentication as same as we described in the previous paragraph. When an end-host joins the group and has the permit to send, it will get the group sender authentication key as well as the group sending key. The sender, receiver, router and server will do the same thing as described in the previous paragraph. And the group sender authentication key update is similar to the sending key update. The only difference to the solution for architecture 2 is that the server registers the group sender authentication key to the IP Authentication Key Management.

The second scheme is that all the senders will send their packets first to the server and then the server does authentication and then multicasts the packet. In this case, only the server needs to have the IP authentication key. So the server just generates an IP authentication key for itself and register it to the IP Authentication Key Management. And the server usually doesn't need to update this key in case that the session key gets updated. The other issues are similar to the ones we described before.

Since we have the active network to check the IP Authentication Header and drop garbage packets, we prevent spamming to the secure multicast group. The protocol for the server to register the entry in the IP Authentication Key Management is still an open issue since there's no specification for the IP Authentication Key Management. But once given the specification of the IP Authentication Key Management, the protocol for the server to register will be feasible.

9 Threat Analysis

In this threat analysis we will identify possible threats and show how they are handled by SMIF.

1. Mallory blocks key update to Alice and then keeps sending her messages with the old session key.
   Solution: Here we clearly have the problem that an unauthorized sender directly forwards messages to Alice. The simple solution to this scenario is to use the sender authorization described in section 7. Because the sender is always authorized in these architectures, Mallory cannot send bogus messages directly if she is not authorized to send. If she is an authorized sender then she could send a message to Alice anyway.
   In the case where all the senders share a common sender key and send the messages directly to the group, the attack becomes dangerous if Mallory was expelled from the authorized senders and she blocks the key update to Alice. Later Mallory still uses her old sender key to directly send forged packets to Alice. To prevent this attack we can use the server based broadcast scheme. Here only the server can send messages to the multicast group and signs each message with his private key.

2. Mallory steals Trent's private key and forwards forged messages to the group members.
   There's no real solution to this kind of attack. Our assumption is that Trent is a fully trusted server and that his private key is not disclosed. Since our architecture relies on this assumption we can not protect ourselves from this attack.

3. Multiple members collaborate to get more keys or joining the group twice to get more keys.
   Solution: This does not help at all since all the keys in the key hierarchy are distinct. This attack can be powerful if we make multiple keys at one level equal which would simplify the key management but also lowers the security. The attack where a receiver shares its key with an outside person is not a valid attack since the receiver can as well share all the multicast information with that person anyway. In the same way when a valid sender shares his sending key with an outsider we don't try to prevent since that fraudulent person can forward packets to the multicast group for the outsider anyway.

Denial of Service Attacks

A denial of service attack stands for a common type of attack where a malicious attacker makes a service unavailable to other users. Denial of service in multicast is much more complicated to prevent than in a pure unicast world because of the following attacks.

1. Spam the multicast group.
   Solution: We can use our approach proposed in section 8 which uses active networks and IPv6 to prevent flooding of the network. In the case where we don't have these protocols we can't prevent flooding. The best we can do is message filtering at the end host based on authentication as described in section **??**.

9.1 Multicast as a Tool of Threat

Can multicast be used as a tool to break down security. Although it is not part of the secure multicast, it is worth mentioning in here. - what happens if a person spam the network with lots of nasty packets - very little packet can result in large flow of packets

10 Conclusion

We have established a general framework which can be used to implement a wide variety of security requirements for multicast groups. Through three example architectures we have shown the usefulness and wide application range of the SMIF framework. In our analysis we motivated the correctness and scalability of the provided solutions.

11 Acknowledgements

We would like to thank Doug Tygar and Hiroaki Kikuchi for their numerous discussions and their helpful comments on security issues.

References

[Atk95] R. Atkinson. IP Authentication Header, RFC 1826. Technical report, IETF, August 1995.

[FJM$^+$95] S. Floyd, V. Jacobson, S. McCanne, C. G. Liu, and L. Zhang. A reliable multicast framework for light-weight sessions and application level framing. In *Proceedings of the ACM SIGCOMM 95*, pages 342–356, Boston, MA, August 1995.

[GR97] Rosario Gennaro and Pankaj Rohatgi. How to Sign Digital Streams. In *CRYPTO*, 1997.

[Mit97] Suvo Mittra. Iolus: A Framework for Scalable Secure Multicasting. In *ACM SIGCOMM*, September 1997.

[Sch96] Bruce Schneier. *Applied Cryptography.* John Wiley & Sons, 1996.

[WGT98] David J. Wetherall, John V. Guttag, and David Tennenhouse. ANTS: A Toolkit for Building and Dynamically Deploying Network Protocols. April 1998.

[WHA97] Debby M. Wallner, Eric J. Harder, and Ryan C. Agee. Key Management for Multicast: Issues and Architectures. Technical report, IETF draft, July 1997.

[XMZY97] X. Rex Xu, Andrew C. Myers, Hui Zhang, and Raj Yavatkar. Resilient Multicast Support for Continuous-Media Applications. In *Proceedings of NOSSDAV 97*, 1997.