CSE 590TU Assignment #5 – Protocols II (revised)

Due at the beginning of class on February 7, 2006

1. Fun with Timestamping

Imagine that you've been asked to digitally sign a document and you want to retain proof both that you signed the document as well as when you signed it. You could simply include a piece of metadata indicating the signing time ("I signed this document Tuesday, 1/31/06, at 5pm.") within the data covered by your signature[1], but since you could have lied about the signing time such statements don't really carry any weight. What you really want is a widely-trusted third party to vouch for you – to say that it saw your signature at some time T. A *timestamping service* provides such services; for a fee, the timestamp service allows clients to send the service hash values, and the timestamping service will sign those hash values together with the current time, producing a *timestamping receipt* for the client's hash value. The timestamping receipt is then sent back to the client, who can do whatever he wants with it (typically, archive it and/or send it along with the signature).

Question 1(a): Assume that you're designing the format of the timestamping receipt – what information would you include in the receipt, why are you including it and what's the minimum size in bytes of the information you have to include? You can assume that hash values sent to you by clients are all SHA2-256 hashes and are thus 32 bytes in size. You can also assume you only need accuracy to the nearest second and that time values are represented as follows:

> GeneralizedTime format: YYYYMMDDhhmmssZ
> Example: "20060131171329Z" is January 31, 2006, 5h13m29s Zulu (GMT)

For any other values you want to include, you may make reasonable assumptions about the sizes of those values so long as you state your assumptions clearly.

After your timestamping service has been up and running for a while, your auditors point out the following concern: if someone malicious can get control of the timestamping server, even if only for a little while, they can effectively create "backdated" timestamps by modifying the machine's notion of time. (That is, a malicious party could cause a timestamp to be issued on February 15 that says it was issued on January 31.) Since each timestamp is independently generated, there would be no way to tell a fraudulently produced "backdated" timestamp and a "genuine" timestamp apart.

Question 1(b): Describe how you could modify the operation of your timestamping service to defend against fraudulent insertion of timestamps "after the fact". What additional information do you have to add to the timestamping receipt to effect this change?

---

[1] For example, if you were using S/MIME to sign a document you could include your statement of the signing time within the Authenticated Attributes section of the PKCS#7/CMS data structure.

2. Encrypted e-mail and mailing lists

In this problem we're going to explore some of the practical difficulties that mailing lists (and mailing list servers) introduce into the S/MIME encrypted e-mail world. In order to do that I first need to give you some background on how S/MIME supports sending encrypted messages to multiple recipients, then talk a little bit about mailing list servers, then we can look at how the two interact.

2.1. Encrypting S/MIME e-mail for multiple recipients

In class we discussed how multiple digital signatures can be attached to a single S/MIME message through the addition of parallel SignerInfos, but we didn't talk about encrypting a message for multiple recipients. For encrypted content, there is a per-recipient structure called RecipientInfo that is used to hold per-recipient key information. Normally it is used as follows: the content is encrypted with a random symmetric key K, and then K is public-key encrypted to each recipient (using that recipient's public key). So, if there are n recipients with public keys PK_1, PK_2, …, PK_n, then the message will have n RecipientInfos where the i^{th} RecipientInfo contains $\{K\}PK_i$, as shown in Figure 1:

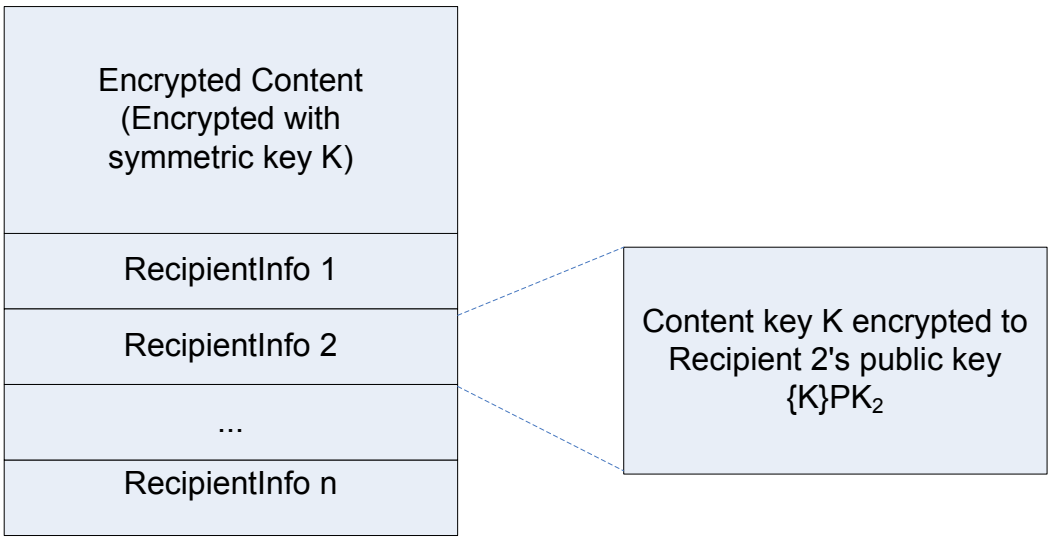


Figure 1: S/MIME Encrypted Content

While we normally use RecipientInfo to store symmetric keys encrypted to recipient public keys, the S/MIME does not require that the per-recipient "wrapping" key be an asymmetric public key. The standard also allows us to use a per-recipient symmetric key to encrypt the content encryption key, which is useful if the sender and recipient have already performed a key exchange. For example, consider a scenario when the sender and recipient know they're going to exchange one S/MIME encrypted e-mail message per minute during a typical day. Instead of performing the cost of a public-key encryption per message, the sender and recipient could perform a Diffie-Hellman key exchange once at the beginning of the day to establish a shared symmetric key K_{Day} and then use that K_{Day} to wrap the per-message symmetric key, as shown in Figure 2:

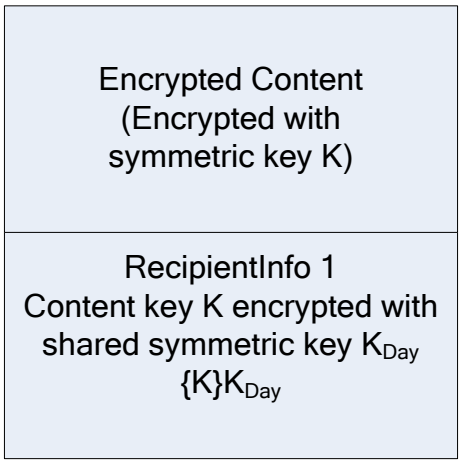


Figure 2: S/MIME encrypted content using a shared symmetric key

2.2. Mailing list servers

You are probably all too familiar with mailing list server like the one that runs the class mailing list[2]. We don't need to get into all the details of how mailing list servers work for the purposes of this problem set, but here are some key points to keep in mind:

1. Mailing list servers may host many mailing lists.
2. The membership of a mailing list may be public, known only to list members, or known only to the mailing list administrator (and the server).
3. The mailing list server controls all additions to and deletions from the mailing list through a single interface. At any point in time the server knows exactly who's supposed to be a member of the list and be able to receive content.
4. When someone wants to send a message to the mailing list, the sender sends the message to an address on the mailing list server for that list. Upon receipt, the mailing list server queues the message for processing. Messages are processed in the order they are received. Due to Internet connectivity delays, mailing list participants may not receive messages in the exact order in which they were sent to the server.
5. Assume that for anti-spam reasons (a) only mailing list subscribers can send mail to the mailing list, and (b) the sender of a message will receive of copy of his sent message back from the server as part of his mailing list subscription.

That's basically all there is to a mailing list server in a nutshell – for the problems that follow if you feel you need more information about the internal operation of a list server send e-mail (preferably to the class list so everyone can see the conversation).

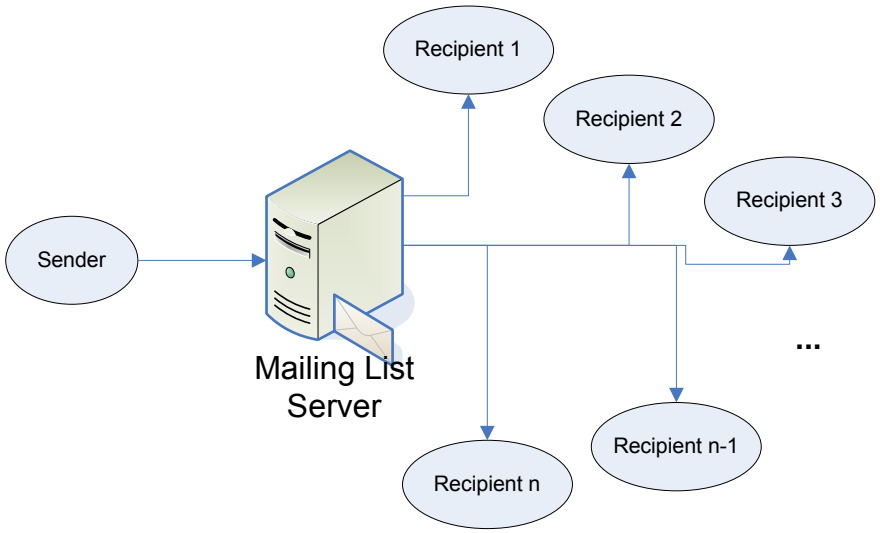
2.3. Adding message encryption to a mailing list

The ultimate goal in this problem is for you to describe how you could add message encryption as a feature to a mailing list. We'll walk through the problem in steps, looking at requirements for senders, recipients and the central mailing list server. Your focus should be on the cryptographic aspects of the problem, particularly the mechanism of message encryption and key distribution. You should not get hung up on operational

[2] The class mailing list server is using an open-source mailing list package called Mailman, which is pretty popular at the moment.

issues like, “What happens if the server runs out of space mid-encryption?” or “What’s the maximum number of users and lists that the list server can handle?”

Let’s start by looking at the message flow:



All of the network links in our scenario are unencrypted (no IPSEC or SSL tunnels exist). The typical scenario begins with the sender composing a message and sending it to the server. The server then re-sends the message to all of the recipients. All message encryption must be done using S/MIME (compliance with standards is a must). Assume that all symmetric encryption is done using the AES block cipher and all public-key encryption is done using RSA.

Operational Questions

Question 2(a): What information does the sender have to know about each mailing list recipient if it wants to be able to send them encrypted messages? What information does each recipient have to know about the mailing list server? If there are M members of the mailing list, how many public keys does each member need to know and how many keys does the server need to know?

Question 2(b): When the sender encrypts a message that he wants to send to the mailing list, how many RecipientInfos will his S/MIME message have? Does it make a difference if the sender wants to archive a copy of his encrypted message in his “Sent Items” folder in case he wants to look at it later?

Question 2(c): When the server receives an inbound message for the mailing list, it first has to verify that the message came from a subscriber to the mailing. This is normally done just by inspecting the From: line of the message, but that’s not secure. Describe one way that the server could verify that the message came from a mailing list subscriber. Does the server need to know any additional information about the sender beyond what you already indicated in you answer to Question 2(a)?

After receiving a message and verifying that it came from a mailing list subscriber, the server now needs to send it to all of the recipients on the mailing list (including the original sender). Assume for the remainder of this problem that the server knows and trusts a public encryption key for each mailing list member:

Question 2(d): If the mailing list has M members, how many public key encryptions does the server have to perform to prepare the message for sending? Does it make a difference if the server prepares one message with many RecipientInfos vs. a separate message for every recipient?

Question 2(e): How much symmetric key decryption and encryption does the server need to do in order to properly relay the message? Assume that the server doesn't keep copies of any messages itself (no mailing list archives, etc.). Justify your answer.

Key Management Questions

Now we're going to look at the key management aspects of implementing the solution you've just sketched. Assume that the mailing list server supports the following operations related to mailing list membership:

1. Subscribe: This operation is called whenever a new user wants to join the mailing list. The user has to provide an e-mail address where he receives mail as part of the subscription process.
2. Unsubscribe: This operation is called whenever a member of the mailing list decides to leave the list.

Additionally, the server has a public encryption key is that well-know and published, so anyone can send an encrypted message to the server.

For the following question, we're going to assume that the mailing list has relatively few subscribers and that not that many messages are sent to the list on a daily basis, so the server can afford to perform a public key encryption per mailing list member for each message sent to the list.

Question 3: During a Subscribe operation, what information does the server need to collect from the new user before added them to the mailing list? Describe a protocol that the new subscribing user and the server can use to send this information to the server and authenticate that it came from the entity that receives e-mail at the subscribing address. [That is, the server needs to know that the subscriber isn't maliciously signing someone else up to the mailing list.]

Now assume that our mailing list in Question 3 has grown very popular, and that it is no longer computationally feasible for the server to perform a public-key encryption per mailing list member for each message sent to the list. The server can only afford to do one public key decryption per message (decrypting the inbound message from the

sender). For encryption, the server will choose a "group symmetric key" that it will use to symmetrically encrypt the content encryption key. Each encrypted message sent by the server will thus now have just one RecipientInfo, containing the content key K encrypted with the current "group symmetric key". The server generates "group symmetric keys" and is responsible for distributing those keys to recipients.

Question 4: Design the "group symmetric key" system. In particular, describe how the server can ensure that new subscribers get the current group symmetric key, how the group symmetric key is used to encrypt a particular message, and what happens to the current group symmetric key when a member unsubscribes from the mailing list. You may assume that the server is capable of performing a public key operation per recipient whenever a new member subscribes to or unsubscribes from the list. (Additionally, if you need to you may make whatever additional reasonable engineering assumptions you wish about the environment so long as you can justify them.)