Design of the TTI Prototype Trusted Mail Agent

Marshall T. Rose†
David J. Farber
Stephen T. Walker

Abstract

The design of the *TTI* prototype Trusted Mail Agent (TMA) is discussed. This agent interfaces between two entities: a key distribution center (KDC) and a user agent (UA). The KDC manages keys for the encryption of text messages, which two subscribers to a key distribution service (KDS) may exchange. The TMA is independent of any underlying message transport system.

Subscribers to the KDC are known by unique identifiers, known as IDs. In addition to distributing keys, the KDC also offers a simple directory lookup service, in which the "real-world" name of a subscriber may be mapped to an ID, or the inverse mapping may be performed.

This document details three software components: first, a prototype key distribution service, which has been running in a TCP/IP environment since December, 1984; second, a prototype trusted mail agent; and, third, modifications to an existing UA, the Rand MH Message Handling system, which permit interaction with the prototype TMA.

† All three authors are with Trusted Technologies, Incorporated, POB 45, Glenwood, MD 21738, USA. Telephone: 301/854–6889. In addition, Professor Farber is with the University of Delaware.

Design of the TTI Prototype Trusted Mail Agent

Introduction

Initially, a brief model of a user community employing a trusted mail service is introduced. Following this introduction, a prototype system is described which attempts to meet the needs of a user community. Finally, open issues are discussed, which are currently not satisfied by the prototype system or its model of operation.

Two or more entities, called *users*, wish to communicate in a *secure* environment. Depending on their available resources, different levels of security are possible. At the extreme, two parties with substantial resources may wish to communicate in a fashion which prevents any third parties, known as *adversaries*, from observing their communication. At this level, not only is an adversary unable to capture the communication for analysis, but in fact, the adversary is unaware that any communication is occurring at all. In most applications, this level of security is prohibitively expensive. A more economic method is to translate messages into a form which is useless to an adversary and then to communicate those messages on an insecure medium.

This latter method requires the two users to have some sort of *key* with which to "lock" the plaintext into ciphertext when transmitting, and then to "unlock" the ciphertext back into useful form when receiving. Hence, there are two central issues to deal with: first, keys must be generated, distributed, and maintained in a secure fashion; and, second, the keys must be "intricate" enough so that sense can't be made out of the ciphertext without knowledge of the key. The first part is handled by a *key distribution center* (KDC), which maintains a list of users and a set of keys for each pair of users. The second part relies on sophisticated encryption and decryption algorithms. It is beyond the scope of this paper to describe cryptographic techniques in detail. For a detailed survey of this area, the reader should consult [VVoyd83].

In the context of our discussion (using the terminology of [X.400]), the medium used to transport is supplied by a *message transport system* (MTS), which is composed of one or more *message transport agents* (MTAs). Usually, the entire MTS is distributed in nature, and not under a single administrative entity; in contrast, an MTA is usually controlled by a single administration and resides in a particular domain. At every end-point in the medium, a *user agent* (UA) acts on behalf of a user and interfaces to a local MTA. This model is briefly summarized in Figure 1.

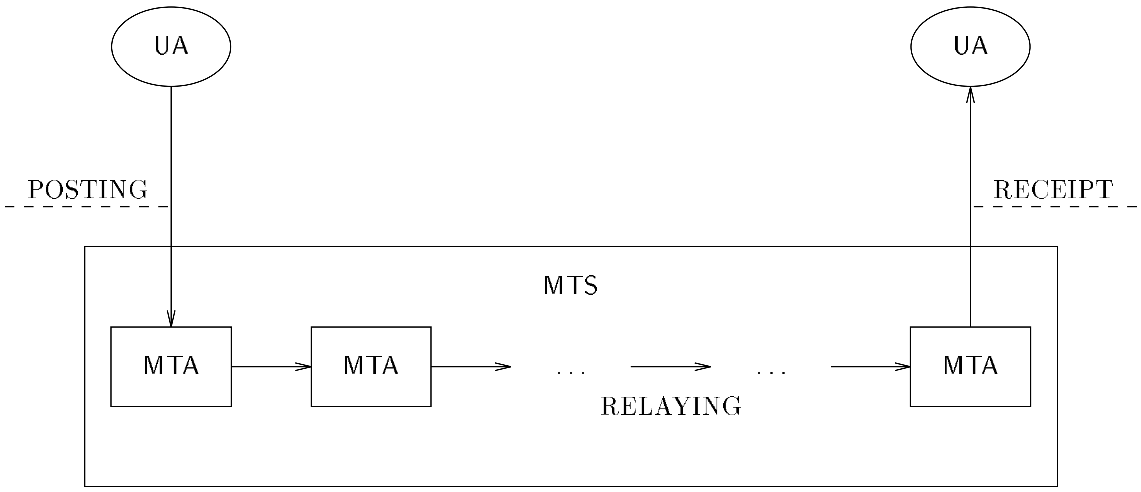

Figure 1
The MTS Model

A message, in our context, consists of two parts: the *headers* and the *body*. The headers are rigorously structured; they contain addressing information and other forms useful to a UA. The body is freely formatted and is usually not meaningful to a UA.

When a message is sent from one user to another, the following activities occur: The originating user indicates to the UA the address of the recipient; the UA then posts the message through a *posting slot* to an MTA, which involves a posting protocol in which the validity of the address and the syntax of the message are considered. Upon successful completion of the protocol, the MTA accepts responsibility for delivering the message, or if delivery fails, to inform the originating user of the failure. The MTA then decides if it can deliver the message directly to the recipient; if so, it delivers the message through a *delivery slot* to the recipient's UA, using a delivery protocol. If not, it contacts an adjacent MTA, closer to the recipient, and negotiates its transfer (using a protocol similar to the posting protocol). This process repeats until an MTA is able to deliver the message, or an MTA determines that the message can't be delivered. In this latter case, a failure notice is sent to the originating user.

It is important to note that there are two mappings which occur here. The first, which is performed implicitly by the originating user, maps the name of the recipient into the recipient's address; the second, which is performed explicitly by the MTS, maps the address of the recipient into a route to get from the originator's MTA to the recipient's MTA. These mappings are depicted in Figure 2.

Obviously, there is no guarantee that the MTS can be made secure, in *any* sense of the word. This is particularly true if it is under several administrations. Regardless of the number of administrations in the MTS, this problem quickly

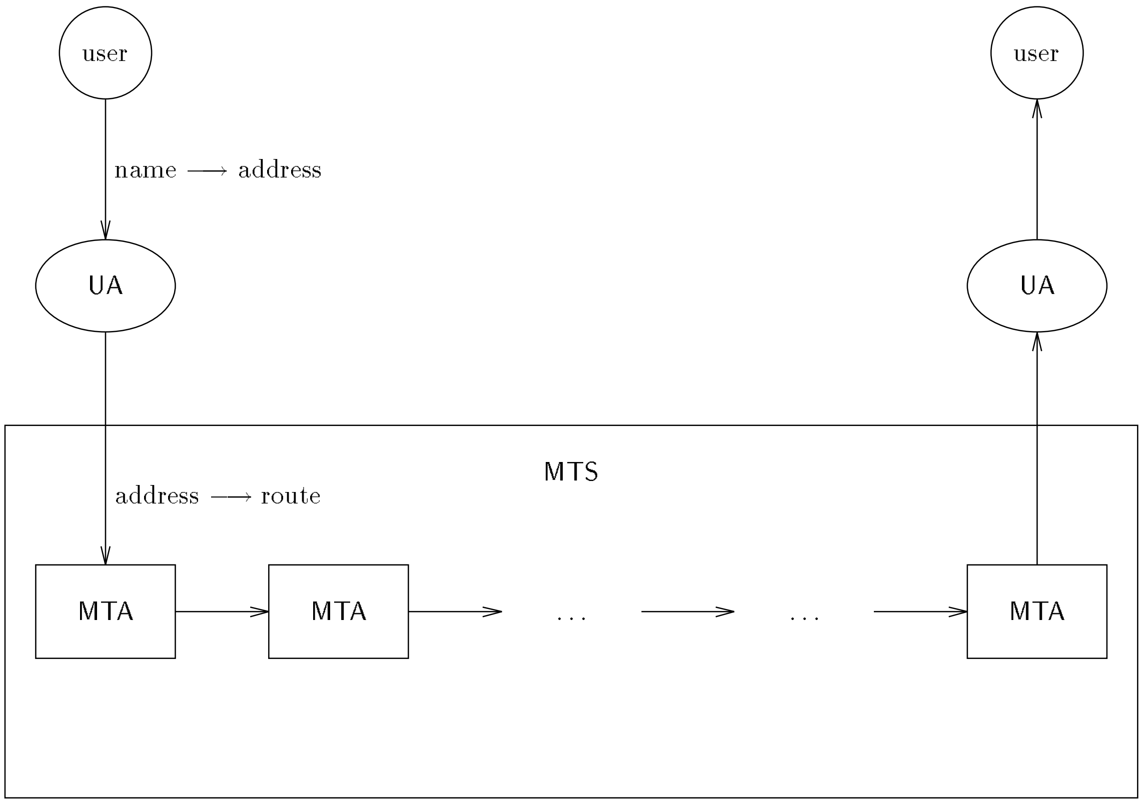

Figure 2
Mappings in the MTS model

degenerates to a problem of Byzantine generals[LLamp82]. Further, trying to secure each MTA in the path that a message travels is equally questionable.

To support secure communications in this environment, a new entity, the *trusted mail agent* (TMA) is introduced into our model. A solution is to have the UA interact with this entity both when posting a message and when taking delivery of a message. The UA first contacts a TMA to encrypt the body of the message for the recipient, prior to pushing it through the posting slot. Upon receipt from the destination MTA, the UA examines the message and contacts the TMA to decipher the body of the message from the source. An overview of the relationship between the standard MTS model and the augmentations made for the Trusted Mail[1] system is shown in Figure 3.

To achieve these tasks, the TMA interacts with a *key distribution service* (KDS), which manages keys between pairwise users. At this point, a third mapping takes place: the UA must be able to map addresses into the identifier(s) by which the originator and recipient are known by the TMA and KDS. These identifiers are known as KDS IDs, or simply IDs. Usually, a fourth mapping also occurs,

[1] Trusted Mail is a trademark of Trusted Technologies, Incorporated.

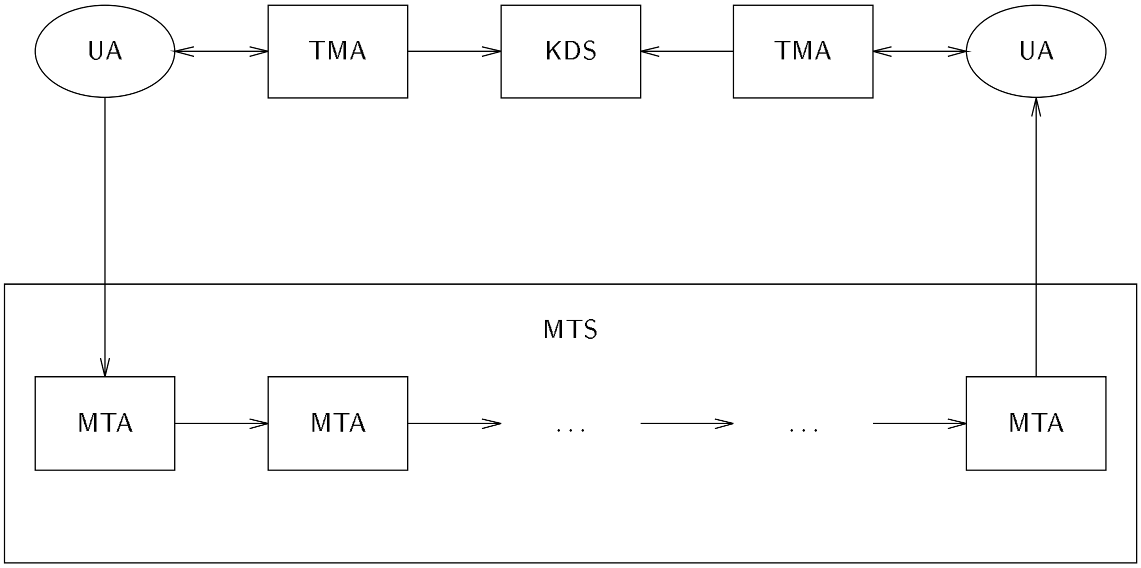

Figure 3
Modifications to the MTS model

which maps the ID of a user into the name of a user. In our context, there is an exact one-to-one mapping between the name of a user and the ID of that user. In contrast, there may be a one-to-many mapping between the name of a user and that user's address in the MTS. Further, there are usually many different routes which a message may traverse when going from an originating user to a recipient user.

The TMA is said to be *trusted* because it can be relied on to perform only those actions specifically requested by the user. In the context of this paper, this means, given proper construction and maintenance of the TMA, that the software will communicate with the KDC in some secure fashion to negotiate key relationships and that it will not disclose those key relationships to other parties. Furthermore, the body of mail messages exchanged between users which employ a trusted mail agent will be unintelligible to other parties. Finally, a recipient of a message receives authenticated information from the trusted mail agent as to the identify of the sender.

Hence, when each user employs a TMA, end-to-end encryption occurs at the UA level (to avoid any problems with malicious MTAs).[2] Any adversary listening in on the MTS, may observe messages, but make no sense out of them (other than rudimentary traffic analysis). Note, however, that since the medium itself is not secure, an adversary may still introduce new messages, corrupt messages, or remove

[2] Note that in the scope of this system, the end-points are the user agents, not the hosts they reside on. In fact, it may very well be the case that the user agent and the local message transport agent do not reside on the same host.

messages, as they traverse the MTS. In the first two cases, however, the recipient would be suspicious because the adversary lacks the encrypting key employed by the source user. In the third case, the source user can retransmit the message after a suitable time. Of course, there is no built-in retransmission policy — this aspect depends on the user's sending mail and is beyond the scope of the system.

It is important to understand the target community for the Trusted Mail system described herein. In particular, the TMA is intended for a commercial and not a military environment. This distinction is important, since it is the *fundamental* assumption of this paper that the latter community has much stricter requirements than the former. Because of this, the prototype system is able to make certain simplifying assumptions which permit it to operate in a mode which is less secure than military applications would permit. Although these issues are explored in greater detail at the end of the paper, for the moment recall that, like most qualities, trustedness is not absolute: there are varying degrees of trustedness, and as a system becomes more trusted, it becomes more expensive, in some sense, to operate and maintain.

It is perhaps instructive at this point to consider why the introduction of a key distribution center is appropriate in this environment, and why the *fundamental* assumption that trusted mail agents do not directly communicate with each other is necessary. Although a user agent is able to converse with the local message transport agent in real-time, it is frequently not able to communicate with other user agents in real-time. Furthermore, considering the vast problems and overhead of trying to establish secure communications from "scratch" (a problem far beyond the scope of this paper), it is would not be a good idea to try and communicate key relationships with other user agents, even if it were always possible to do so. In addition, by separating the trusted aspects of the message transport system from the system itself, many other advantages can be seen. These are presented in greater detail at the end of the paper.

The discussion thus far has considered only a single recipient. In practice, a user might wish to send to several others, using a different key for each. Hence each copy of the message is encrypted differently, depending on the particular recipient in question. Note that this has the effect of *un-bundling* message transfer in the MTS, as advanced MTAs tend to keep only a single copy of the message for any number of recipients in order to save both cpu, disk, and I/O resources.

For example, in some existing mail systems, if a message was sent to n users on a remote system, then the n addresses would be sent from the source MTA to the remote MTA along with one copy of the message. Upon delivery, the remote MTA would deliver a copy to each of the n recipients, but the virtual wire between the source MTA and the recipient MTA was burdened with only one copy of the message. But in a secure environment, since a different key is used by the source user when communicating with each of the n recipients, n different messages will be posted with the local MTA, and the advantages of recipient bundling are lost.

Along these lines however, private discussion groups may wish to avoid this problem by establishing access to a single ID for their use. In this case, a subscriber to the KDS may actually have more than one ID, one for “personal” use and one for each discussion group. The appropriate ID is used when posting messages to the discussion group. Naturally the administrative policy for deciding who is allowed to use the KDS ID of a discussion group is left to the moderator of the group. Observant readers will note that this vastly decreases the aspect of secure communications for the discussion group. This method is suggested as a compromise which permits the bundling of messages for multiple recipients to reduce MTS traffic. The price is high however, as a compromise on behalf of *any* member of the discussion group compromises the entire group. For large discussion groups and a bandwidth limited MTS, this price may be worth paying. The prototype implementation of the TMA supports multiple recipients but not multiple KDS IDs.

Having described this environment for communication, the designs of a KDS and TMA which form the heart of the *TTI* Trusted Mail system are discussed. The prototype system was developed on a VAX[3]-11/780 running 4.2BSD UNIX[4]. The system is based on the ANSI draft[FIKM] for financial key management, but diverges somewhat in operation owing to the differences between the electronic mail (CBMS) and electronic funds (EFT) environments. Note however that the ANSI data encryption algorithm[DEA, FIPS46] is used in the current implementation. A public-key cipher system was not considered as the basis for the prototype since, to the authors' knowledge, an open standard for a public-key system has yet to be adopted by the commercial community. In contrast, the ANSI draft for financial key management appears to be receiving wide support from the commercial community.

In the description that follows, a large number of acronyms are employed to denote commonly used terms. In order to aid the reader, these are summarized in Table 1.

The Key Distribution Service

The prototype version of the KDS was designed to provide key distribution services for user agents under both the same or different administrations. As a result, the means by which a trusted mail agent connects to a key distribution server is quite flexible. For example, the prototype system supports connections via standard terminal lines, dial-ups (e.g., over a toll-free 800 number), UNIX pipes, and over TCP sockets[IP, TCP]. In the interests of simplicity, for the remainder of this paper, a TCP/IP model of communication is used. Initially, a server on a

[3] VAX is a trademark of Digital Equipment Corporation.
[4] UNIX is a trademark of AT&T Bell Laboratories.

Abbrev.	Term	Context
CBC	Cipher Block Chaining	DES
CBMS	Computer Based Message System	
CKD	Key Distribution Center	EFT
CKS	Checksumming	DES
CSM	Cryptographic Service Message	
DEA	Data Encryption Algorithm	
DES	Data Encryption Standard	
DSM	Disconnect Service Message	MCL
ECB	Electronic Code Book	DES
EFT	Electronic Funds Transfer	
IDK	Key Identifier	CSM
ID	Identifier	KDS
IP	Internet Protocol	
IV	Initialization Vector	CSM
KA	Authentication Key	CSM
KDC	Key Distribution Center	CBMS
KDS	Key Distribution Server	CBMS
KD	Data-encrypting Key	CSM
KK	Key-encrypting Key	CSM
MAC	Message Authentication Code	CSM
MCL	Message Class	CSM
MH	The Rand Message Handling System	
MIC	Message Integrity Code	CSM
MK	Master Key	CSM
MTA	Message Transport Agent	CBMS
MTS	Message Transport System	CBMS
ORG	Message Originator	CSM
RCV	Message Receiver	CSM
RIU	Request Identified User	MCL
RSI	Request Service Initialization	MCL
RUI	Request User Identification	MCL
TCP	Transmission Control Protocol	
TMA	Trusted Mail Agent	CBMS
TTI	Trusted Technologies, Inc.	
UA	User Agent	CBMS

Table 1
Abbreviations used in this paper

well-known service host in the ARPA Internet community listens for connections on a well-known port.[5] As each connection is established, it services one or more transactions over the lifetime of the session. When all transactions for a session have been made, the connection is closed. If the necessary locking operations are performed by the server to avoid the usual database problems, then more than one connection may be in progress simultaneously. Of course, a time-out facility should

[5] The term *well known* in this context means that the location of the service is known *a priori* to the clients.

also be employed to prevent a rogue agent from monopolizing the key distribution server.

Once a session has been started, the client (a.k.a. TMA) initiates transactions with the server (a.k.a. KDS). Each transaction consists of the exchange of two or three *cryptographic service messages* (CSMs): the client sends a request, the server attempts to honor the request and sends a response, and, if the server responded positively, the client then acknowledges the transaction. By exchanging these cryptographic service messages, the KDS and the TMA are able to communicate key relationships. Obviously, the relationships themselves must be transmitted in encrypted form.[6] Hence, not only are key relationships between two TMAs communicated, but key relationships between the KDS and the TMA are communicated as well.

This leads us to consider the key relationships that exist between a TMA and the KDS. A client usually has three keys dedicated for use with the server. The first is the *master key* (denoted MK), which has an infinite cryptoperiod, and is rarely used. This key is distributed manually. The second is the *key-encrypting key* (denoted KK), which has a shorter cryptoperiod. Whenever a KK is transmitted to the TMA, it is encrypted with the master key. The third is the *authentication key* (denoted KA), which is used to authenticate transactions that do not contain data keys (a count field is also used to avoid play-back attacks). Whenever a KA is transmitted to the TMA, it is encrypted with the key-encrypting key. When transactions contain keys, an associated count field is included to indicate the number of keys encrypted with the key-encrypting key used. Although not used by the prototype implementation, a production system would employ audit mechanisms to monitor usage histories.

Currently four types of requests are honored by the KDS: two key relationship primitives, and two name service primitives. The type is indicated by the *message class* (MCL) of the first cryptographic service message sent in the transaction. As each message class is discussed, the appropriate datastructures used by the KDS are introduced. Space considerations prevent a detailed description of the information exchanged in each transaction. Appendix B of this paper presents a short example of an interaction between the KDS and a TMA.

The first two requests are used to create (or retrieve) key relationships, and to destroy key relationships:

The *request service initialization* (RSI) message class is used to establish a *key-encrypting key* (KK) relationship between the TMA and another TMA, or between the TMA and the KDS. As implied by the name, a key-encrypting key is

[6] Otherwise an adversary could simply impersonate a TMA and ask for the desired key relationships. Similarly, this also prevents an adversary from successfully impersonating a key distribution server.

used to cipher keys which are used to cipher data exchanged between peers. These other keys are called *data keys* (KDs).

The *disconnect service message* (DSM) message class is used to discontinue a KK-relationship between the TMA and another TMA, or between the TMA and the KDS. This prevents keys which are felt to have been compromised, or are vulnerable to compromise, from receiving further use in the system. It should be noted that, owing to mail messages (not CSMs) in transit, a discontinued key relationship may be needed to decipher the key used to encipher a mail message. The prototype KDS supports this capability.

In addition to maintaining an MK/KK/KA triple for each TMA, the KDS also remembers KK-relationships between TMAs. The reason for this stems from a fundamental difference between the electronic funds transfer and computer-based message system worlds. The KDS assumes that no two arbitrarily chosen TMAs can communicate in real-time, and as a result, TMAs do not exchange cryptographic service messages. (See Appendix C for a more detailed discussion.) This means that when a TMA establishes a KK-relationship with another TMA, the former TMA may start using the KK before the latter TMA knows of the new KK-relationship. In fact, it is quite possible for a KK-relationship to be established, used, and then discontinued, all unilaterally on the part of one TMA. It is up to the KDS to retain old cryptographic material (possibly for an indefinite period of time), and aid the latter TMA in reconstructing KK-relationships as the need arises. Naturally, discontinued KKs are not used to encode any new information, but rather to decode old information. (Again, refer to Appendix C for additional details.)

The other two requests are used to query the directory service aspects of the key distribution server:

The *request user identification* (RUI) message class is used to identify a subscriber to the KDS. Both the KDS and TMA are independent of any underlying mail transport system (MTS). As a result, a subscriber to the KDS is known by two unique attributes: a "real-world" name, and a KDS identifier (ID). The user of a mail system, or the UA, is responsible for mapping an MTS-specific address (e.g., `MRose@UDEL.ARPA`) to the person associated with that maildrop (e.g., `''Marshall␣T.␣Rose''`). When conversing with the KDS, the TMA uses the KDS ID of another user to reference that person's TMA. Since it is inconvenient to remember the IDs (as opposed to people's names), the KDS provides the RUI message class to permit a TMA to query the mapping between names and IDs. If the KDS cannot return an exact match, it may respond with a list of possible matches (if the identifying information given was ambiguous), or it may respond with a response that there is no matching user.

Finally, the *request identified user* (RIU) message class performs the inverse operation: given a KDS ID, a "real-world" name is returned. This request is useful for disambiguating unsuccessful RUI requests and in boot-strapping a TMA.

The KDS maintains two directories: a private directory and a public directory. The private directory contains all information on all clients to the KDS. The public directory is a subset of this, and is used by the KDS when processing RUI and RIU requests.[7] As a result, certain clients of the KDS may have unlisted IDs and names.

The Trusted Mail Agent

The prototype version of the TMA was designed to interface directly to the user agent in order to maximize transparency to the user. In present form, the TMA is available as a load-time library under 4.2BSD UNIX, although efforts are currently underway to transport the TMA to a PC-based environment.

The software modules which compose the TMA contain a rich set of interfaces to the KDS. In addition, the TMA manages a local database, so responses from the KDS may be cached and used at a later time. In all cases, the KDS is consulted only if the information is not present in the TMA database, or if the information in question has expired (e.g., KK-relationships). This caching activity minimizes connections to the KDS. Although connections are relatively cheap in the ARPA Internet, substantial savings are achieved for PCs which contact the KDS over a public phone network (dial-up) connection.

The TMA performs mappings between pairs of the following objects: user names, KDS IDs, and MTS addresses. The TMA considers all trusted mail agents, including itself, as a user name, KDS ID, and one or more MTS addresses. Although the TMA does not interpret addresses itself, in order to simplify mail handling, the TMA remembers the relationship between these objects so the user enters this information only once.

Initially, when a TMA is booted, the user supplies it with the master key and the user's KDS ID. Both of these quantities are assigned by the personnel at the key distribution center, and subsequently transmitted to the user via an alternate, bonded service.[8] The TMA connects with the KDS and verifies its identity. From this point on, the TMA manages its KK-relationships between the KDS and other TMAs without user intervention.

The current implementation of the TMA assumes a "general memo framework" in the context of the Standards for ARPA Internet Text Messages[DCroc82]:

[7] In the prototype implementation, the two directories are, for the moment, identical.
[8] In this fashion, the problems of boot-strapping over an unsecure medium are avoided.

1. A message consists of two parts: the *headers* and the *body*. A blank line separates the headers from the body.
2. Each (virtual) line in the headers consists of a keyword/value pair, in which the keyword is separated from the value by a colon (:). The headers are rigorously structured in the sense that they contain addressing and other information useful to a user agent.
3. The body is freely formatted and must not be meaningful to a user agent. However, as will be seen momentarily, the body of encrypted messages must have an initial fixed format which the TMA enforces.

This format is widely called "822" after the number assigned to the defining report[DCroc82].[9]

To support the cipher activities described below, the TMA contains internal routines to perform the following DES functions: electronic code book (ECB) for key encryption, cipher block chaining (CBC) for mail message encryption, checksumming (CKS) for mail message and CSM authentication. Readers interested in these different modes of operation for the DES should consult [FIPS81].

Encrypting Mail

To encipher a message, the method used is a straightforward adaptation of the standard encrypting/authentication techniques (though the terminology is tedious). Consider the following notation:

$a_x(s)$: the checksum of the string s using the key x (DEA *checksumming* authentication)

$a_{x+y}(s)$: the checksum of the string s using the exclusive-or of the two keys x and y

$e_x(y)$: the encryption of the key y using the key x (DEA *electronic code book* encryption)

$e_{x,y}(s)$: the encryption of the string s using the key x and initialization vector y (DEA *cipher block chaining* encryption)

h: the headers of the message

and,

b: the body of the message

[9] Although an 822–style framework is employed by the TMA prototype, the 822 ``Encrypted:'' header is not currently present in encrypted messages. This is due to a design decision which assumes that nothing in the headers of a message is sacred to the transport system, and that "helpful" munging might occur at any time. In the real world, such helpfulness is often a problem.

For each message to be encrypted, a data key, initialization vector, authentication key (KD/IV/KA) triple is generated by a random process. (It goes without saying that the integrity of the system depends on the process being *random*). Then, for each user to receive a copy of the encrypted message, the following actions are taken:

First, the headers of the message are output in the clear. Then, a *banner* string, i, is constructed and placed at the beginning of the body of the message:

```
ENCRYPTED MESSAGE: TTI TMA
```

which identifies the message as being encrypted by the *TTI* TMA. Following the banner string is a structure, m, which takes on the syntax and most of the semantics of a cryptographic service message:

MCL/	MAIL
RCV/	rcvid
ORG/	orgid
IDK/	kkid
KD/	$e_{kk}(ka)$
KD/	$e_{kk}(kd)$
IV/	$e_{kd}(iv)$
MIC/	$a_{ka}(b)$
MAC/	$a_{kd+ka}(m)$

After this, the encrypted body is output, $e_{kd,iv}(b)$. In short, the entire output consists of

$$h + i + m + e_{kd,iv}(b).$$

The purpose of the structure m is many-fold. The MCL field indicates the structure m's type; currently only the type MAIL is generated and understood. The RCV and ORG fields identify the intended recipient of the message and the originator. The IDK field identifies the key-encrypting key, KK, used to encrypt the next two fields. The first KD field has the encrypted authentication key, KA, used to calculate the MIC of the plaintext of the body of the message. After the body of the message is deciphered, $a_{ka}(b)$ is calculated and compared to the value of the MIC field. Hence, the MIC field authenticates the message body. The second KD field has the encrypted data encrypting key, KD, which along with the encrypted initialization vector in the IV field was used to generate the ciphertext of the body. Finally, the MAC field authenticates the m structure itself. The use of a data key, initialization vector, authentication key (KD/IV/KA) triple permits us to perform key distribution in a hierarchical fashion and allows the system to use a KK-relationship over a longer cryptoperiod without fear of compromise.

The TMA provides three primary interfaces to a UA to send encrypted mail: the first takes a file-descriptor to a message and returns a structure g (called a *group*) describing the ciphertext version of the body (this structure contains a KD, IV, and KA generated at random, along with a file-descriptor to the plaintext headers, a file-descriptor to the ciphertext body, and the checksum of the plaintext body); the second takes a user entry (or MTS address) and g, and returns a file-descriptor to the encrypted message for that user (or MTS address); the third takes g and performs clean-up operations. The chief advantage to this scheme of encryption is that if the message is to be sent to more than one recipient, then the MIC and the encrypted body need only be calculated once, since the KD, IV, and KA remain constant (only the KK's change with each recipient, hence for each copy of the encrypted message, only the structure m need be re-calculated).

There are, however, a few subtleties involved: <u>first</u>, the MTS usually accepts only 7–bit characters, so the encrypted text is exploded to consist of only printable characters;[10] <u>second</u>, since the MTS may impose limits on the length of a line, each line of output is limited to 64 characters; and, <u>third</u>, since the body may require trailing padding, during encryption one last unit of 8 bytes is written (and encrypted), naming the number of characters (presently, nulls) padded in the previous 8 bytes (0… 7).

Decrypting Mail

To decipher a message, the method is also straightforward: The headers are output in the clear. The banner string is essentially ignored, and the structure m is consulted to identify the correct key-encrypting key. The TMA checks to see if it knows of that KK. If not, it asks the KDS to supply it. From that point, the KA, KD, and IV are deciphered. The m structure is then authenticated. With the correct key, the remainder of the body is deciphered, and all except for the last 16 bytes are output. The last 8 bytes indicate how many of the previous 8 bytes should be output. So, the appropriate number of bytes is output, and the plaintext body is authenticated and compared to the MIC. Needless to say, as the body is deciphered, it is imploded back to 8–bit characters and lines are restored to their previous lengths. To indicate that the message was correctly deciphered, a new header of the form

```
X-KDS-ID: orgid (originator's name)
```

is appended to the headers of the message. Note that this provides an authentication mechanism. Note, further, that the UA did not have to know the identity of the sender of the message.

---

[10] As a rule, in all CSMs, when encrypted information is transmitted, it is exploded after encryption by the sender, and imploded prior to decryption by the receiver.

Modifications to MH

MH is a public domain UA for UNIX, which is widely used in dealing with both a large number of electronic mail application and a large number of messages. Although this document does not intend to describe MH, parts of the system are described as they relate to the TMA. Readers interested in MH should consult either the user's manual[MRose85a] for a detailed description, or [MRose85d] for a higher-level description.

To modify MH in order to make use of a TMA, three programs were changed (with a high degree of transparency to the user), and two new programs were introduced.

In MH, when a user wishes to send a composed draft (which may be an entirely new message, a re-distribution of a message, a forwarding of messages, or a reply to a message), the user invokes the *send* program. This program performs some minor front-end work for a program called *post* which actually interacts with the MTS. A new option to the *send* and *post* programs, the '`-encrypt`' switch, is introduced. If the user indicates

```
send␣-encrypt
```

then *post* encrypts the messages it sends.

When sending an encrypted message, *post* first checks that each addressee has a mapping to a KDS ID during address verification. Then, instead of batching all addresses for a message in a single posting transaction, for each addressee, *post* consults the TMA for the appropriately encrypted text and posts that instead. (Appendix A discusses the reasons for this more fully.) Hence, assuming the user has established mappings between MTS addresses and KDS IDs, the TMA does all the work necessary to encrypt the message, including contacting the KDS as necessary.[11]

In MH, when a user is notified that new mail has arrived, the *inc* program is run. As each message is incorporated into the user's message handling area, a scan (one-line) listing of the message is generated.

By default, the *inc* program upon detecting one or more encrypted messages, after the scanning process, asks the TMA to decipher the message, and if successful, scans the deciphered messages. This action can be inhibited with the '`-nodecrypt`' switch. Hence, if the user wishes to retain messages in encrypted form, *inc* can be told to note the presence of encrypted messages, but otherwise not to process them. By using the MH user profile mechanism, *inc* can be easily customized to

[11] Once the TMA establishes a connection to the KDS, it retains that connection until the UA terminates. This is done to minimize connections to the KDS. In the context of MH, since the trusted mail agent is active over the lifetime of an invocation of a program such as *post*, this means that the connection is terminated just before the program terminates.

reflect the user's tastes. Again, the actions of the TMA are transparent to the user. In fact, if encrypted mail is received from users unknown to the TMA, it queries the KDS as to their identity prior to retrieving the KK-relationship.

If *inc* fails to decrypt a message for some reason, or if *inc* was told not to decrypt a message, the *decipher* program can be used. This simple program merely deciphers each message given in its argument list. The *decipher* program can be given the '`-insitu`' switch, which directs it to replace the ciphertext version of the message with the plaintext version; or, the '`-noinsitu`' switch can be used indicating that the ciphertext version of the message should be left untouched and the plaintext version should be listed on the standard output.

Finally, the *tma* program is used to manipulate the TMA database, containing commands to boot the database, add new users to the database, and to establish mappings between addresses and users in the TMA database. This program can also be used to disconnect KKs between other TMAs, and the KK/KA between itself and the KDS.

Appendix A of this paper contains a transcript of an MH session.

Remarks

We now consider the merit of the system described. After presenting some of the basic strengths of the system and a few unresolved questions, the discussion centers on the simplifying assumptions made by the system, and how these can be defended in a non-military environment.

Strengths

It can be argued that the prototype system (and the augmented model in which it finds its basis) present many strengths.

Perhaps the most important is the high-level of independence from the MTS enjoyed by the system. As a result, since the TMA does not interact directly with the MTS, it can be made to be completely free from any MTS-specific attributes, such as naming, addressing, and routing conventions. Furthermore, when interfacing a Trusted Mail system, no modifications need be made to the MTS or local MTA.

In addition to the systems-level advantage to this scheme, users of the system profit as well, since many disjoint MTSs can be employed by a user with a single TMA. This reduces the number of weaknesses in the system and allows a user to keep a single database of "trusted" correspondents. It should also make analysis and verification of the TMA easier.

Of course from the user-viewpoint, once the TMA has been initially booted, all key management is automatic. Not only does this reduce the risk of compromise of cryptographic material (given proper construction and maintenance of the TMA), but it relieves the user of a tedious and error-prone task.

Finally, although the KDS described herein is used to support Trusted Mail, other applications which require key management, could employ the services offered by the key distribution center.

Open Questions

At present, there are many restrictions on the prototype implementation described. Some of these result from that fact that the implementation is a prototype and not a production system. Others deal with more fundamental issues.

In terms of the TMA, the expiration delay for keys is hard-wired in; it should be user-settable. In the prototype version, the KK and KA with the KDS are good for 2 days or 10 uses (whichever comes first), while a KK for use with another TMA is good for 1 day or 5 uses. In actual practice, keys with long cryptoperiods might be good for 6 months or 100 uses, while keys with short cryptoperiods might be good for 1 month or 25 uses. The choice of actual values is an open question beyond the scope of prototype system.[12] In many respects, this issue is a classic trade-off: with relatively small cryptoperiods, an adversary has less chance of breaking a key, but with longer cryptoperiods less connections have to be made to the key distribution server.

A fundamental issue, owing to differences between the EFT and CBMS environments, is that the KDS implements only a subset of the ANSI draft and the semantics of certain operations have changed somewhat. It would be nice to unify the CBMS and EFT views of a *key distribution center* (in the former environment, the center is called a KDC, while in the latter environment, the center is known as a CKD). Appendix C of this paper discusses the differences between the two perspectives in greater detail.

At present, the relationship between errors in the TMA and the posting process is an open question. For example, if an address doesn't have a mapping in the TMA database, *post* treats this as an address verification error. This prevents the draft from being posted. The philosophy of the UA is unclear at this point, with respect to how recovery should occur. A second area, also in question, deals with the way in which plaintext and ciphertext versions of a message are present in a system. Clearly, it is a bad idea to make both versions available, but since the TMA doesn't try to concern itself with first party observation, there seems to be little possibility of preventing this behavior. The best that can be done, at this stage, is simply to choose a consistent policy that user's should attempt to adhere

[12] The current values were chosen by guess work. Although not necessarily technically sound, the small numbers were very good for debugging purposes.

to. The software can help somewhat in implementing this policy, but it certainly can't circumvent the user.

The prototype is built on the assumption that a single key distribution server is present. Since the ANSI draft[FIKM] makes provisions for *key translation centers*, the Trusted Mail prototype should perhaps be made to operate in a more diverse environment. Until the issues become clearer, this remains open.

Finally, for distribution lists, a large number of people would need to share the same KDS ID. The current implementation doesn't support this. Each TMA database is for a particular ID. A user with multiple IDs would need multiple databases, or the database should be re-organized.

Weaknesses

As pointed out earlier, this prototype system situates itself in a commercial, not military, environment. With respect to this decision, several aspects of the system are now discussed, which we feel are acceptable in a commercial environment, but which would be considered weaknesses in a military environment:

1. Traffic Flow
   The prototype TMA makes no attempt whatsoever to prevent or confuse traffic analysis by augmenting traffic flow.

2. The Database of KDS Subscribers
   Since information returned by the request user identification (RUI) and request identified user (RIU) MCLs are returned in the clear, this allows an adversary to ascertain subscribers to the KDS, and perhaps deduce some information about the system. Without knowledge of the master key however, an adversary could not impersonate a subscriber though. Still, in the military sense, this is a weakness. However, all this assumes that the database maintained by the KDS accurately reflects the real-world.

3. Multiple Recipients
   It is possible, though not proven to the authors' knowledge, that the scheme used to avoid encrypting the body of a message more than once for multiple recipients might permit one of the recipients who is also an adversary to compromise the key relationship between the sender and another recipient.

   The scenario goes like this: When a message is being prepared for encryption, a single KD/IV/KA triple is generated to encrypt the body. Since the sender has a different key relationship with each recipient, each message sent is different, since the structure m depends not only on the KD/IV/KA triple but also on the key relation between the sender and a particular recipient. Now suppose that one of the recipients, r_1, in addition to receiving the copy of the message meant for him/her also intercepts a copy of the message destined for another recipient, r_2. At this point, the recipient r_1 has both the plaintext and ciphertext version of the body, the plaintext version of the KD/IV/KA triple, and the ciphertext version of the KD/IV/KA triple that was generated using the key relationship between the sender and the recipient r_2. The question is: can r_1 now deduce the key relationship between the sender and r_2?

If so, then the way that the TMA attempts to minimize the use of encryption resources is a weakness. But, even if this is possible, given relatively short cryptoperiods for key relationships between TMA peers, this becomes a non-problem.

4. Discussion Groups
As discussed earlier, the proposed method of associating a single KDS ID with the membership of a discussion group does introduce a significant weakness for the security of messages sent to the discussion group. Since the TMA does not assume a general broadcast facility, it appears that there are no good solutions to the problem of discussion group traffic. Of course, it is easy enough to simply send to each member of the group.

   For the sake of argument, let's assume that the discussion group has n members. Now, since a different key relationship would exist between the sender and each of the n recipients, the structure m would be different for each recipient and so a different message would have to be sent to each recipient. To make matters worse, if one rejects the way the TMA handles multiple recipients, not only does the MTS get burdened with n different messages, but the sender's TMA gets burdened by having to encrypt the body of the message n times. For meaningful values of n (say on the order of 500, or even 25), the amount of resources required for any trusted discussion group are simply too costly.

Compromises, Compromises

Each of the possible weaknesses discussed above represent a compromise between the expense of the system and the level of security it can provide.

The first two areas, if addressed by the TMA, could result in much less background information being available to an adversary. In an application where it is important that an adversary not know who is talking to whom, or who can talk at all, this is very important. It is the authors' position that in the commercial environment, this issue is not paramount. By ignoring the issue of traffic flow, the TMA has a lot less work to do and the MTS is kept clear of "useless" messages. By keeping the information returned by the RUI and RIU MCLs in the clear, the complexity of the TMA is significantly reduced.

The second two areas, if addressed by the TMA, could result in a lesser probability of traffic being deciphered by an adversary. Regardless of the application, this is always extremely important. However, the authors' feel that the compromise made by the TMA in these two issues is not substantial, and does not result in an explicit weakness when a message is sent to multiple recipients (note that when there is only a single recipient of a message, these two policies can not introduce weaknesses). In return, efficient use can be made of both the MTS and the TMA when messages are being sent to multiple recipients. Given scarce resources or large numbers of recipients, this approach may prove to be quite winning.

Of course, much work remains to be done to prove the success of the TMA in all four of these areas.

Acknowledgements

The prototype implementation described herein utilizes a public domain implementation of the DES algorithm[DEA] which was originally implemented by James J. Gillogly in May, 1977 (who at that time was with the Rand Corporation, and is now affiliated with Gillogly Software). Interfaces to Dr. Gillogly's implementation were subsequently coded by Richard W. Outerbridge in September, 1984 (who at that time was with the Computer Systems Research Institute at the University of Toronto, and is now affiliated with Perle Systems, Incorporated).

The authors would like to acknowledge Dennis Branstad, Elaine Barker, and David Balensen of the National Bureau of Standards for their comments on the prototype system and insights on the ANSI draft[FIKM]. In particular, Dr. Branstad originally suggested the method used for encrypting a single message for multiple recipients under different keys.

The authors (and all those who have read this paper) would like to thank Willis H. Ware of the Rand Corporation, and Jonathon B. Postel of the USC/Information Sciences Institute. Their extensive comments resulted in a much more readable paper. In addition, the authors would like to thank Dr. Stephen P. Smith and Major Douglas A. Brothers for their insightful comments.

References

[DCroc82] D.H. Crocker. Standard for the Format of ARPA Internet Text Messages. Request for Comments 822. ARPA Internet Network Information Center (NIC), SRI International (August, 1982).

[DEA] *Data Encryption Algorithm*, X3.92–1981, American National Standards Institute, 1981.

[FIKM] *Financial Institution Key Management*, X9.17–198_ (draft), American National Standards Institute, 198_.

[FIPS46] *Data Encryption Standard*, Federal Information Processing Standards, Publication 46, 1977.

[FIPS81] *DES Modes of Operation*, Federal Information Processing Standards, Publication 81, 1980.

[IP] Internet Protocol. Request for Comments 791 (MILSTD 1777). Appearing in *Internet Protocol Transition Workbook*, ARPA Internet Network Information Center (NIC), SRI International, 1981.

[LLamp82] L. Lamport, R. Shostak, M. Pease. The Byzantine Generals Problem. *ACM Transactions on Programming Languages and Systems 4* (July, 1982), 382–401.

[MRose85a] M.T. Rose, J.L. Romine. The Rand MH Message Handling System: User's Manual. UCI Version. Department of Information and Computer Science, University of California, Irvine (January, 1985).

[MRose85d] M.T. Rose, E.A. Stefferud, J.N. Sweet. MH: A Multifarious User Agent. *Computer Networks* (to appear).

[TCP] Transmission Control Protocol. Request for Comments 793 (MILSTD 1778). Appearing in *Internet Protocol Transition Workbook*, ARPA Internet Network Information Center (NIC), SRI International, 1981.

[VVoyd83] V.L. Voydock, S.T. Kent. Security Mechanisms in High-Level Network Protocols. *Computing Surveys 15*, 2 (June, 1983), 135–171.

[X.400] *Message Handling Systems: System Model-Service Elements*, Recommendation X.400, International Telegraph and Telephone Consultative Committee (CCITT).

```
1  % tma -add -user "UCI Portal" uci@udel-dewey
2  3: "UCI Portal"
3      uci@udel-dewey
4
5  % comp
6  To: uci
7  Fcc: +outbox
8  Subject: test message
9  --------
10 mumble, mumble.
11 ^D
12
13 What now? send -encrypt
14  -- Address Verification --
15   -- Local Recipients --
16   uci: address ok
17  -- Address Verification Successful --
18  -- Posting for All Recipients --
19   -- Local Recipients --
20   uci: address ok
21  -- Recipient Copies Posted --
22  -- Filing Folder Copies --
23   Fcc outbox: folder ok
24  -- Folder Copies Filed --
25 Message Processed
```

Figure 4
Sending Encrypted Mail

Appendix A: An MH Session

In the following, the user ‘‘Marshall␣T.␣Rose’’ logged onto host ‘‘udel-dewey’’, wishes to send a message to a user known as the ‘‘UCI␣Portal’’ (a system maintenance account). As shown in Figure 4, line 1, the user first establishes a mapping between the name ‘‘UCI␣Portal’’ and the address uci@udel-dewey. Once this mapping is performed, it remains in effect until the user indicates otherwise to the TMA. When the *tma* program is invoked, it consults the TMA database to see if that user is known. If not, it contacts the KDS to ask for the KDS ID associated with the user. If the response is successful (in this case, the KDS ID is ‘‘3’’), then the TMA updates its database. The *tma* program indicates in its output the KDS ID associated with the user, along with all known addresses (in this case, only one). So, once the name to address mapping has been described the user, the user agent, MH, deals only with the address, while the trusted mail agent deals with the name and KDS ID aspects of the user.

Next, the *comp* program is invoked to compose a new draft on line 5. The user addresses the local user ‘‘uci’’ in the To: field, and indicates that a plaintext copy should be kept in the folder ‘‘+outbox’’. After entering the subject and text of the draft, the user enters What now? level on line 13. At this point, the

```
% inc
Incorporating new mail into inbox...

   1+E02/28 0227-EST mrose             test message  <<ENCRYPTED MESSAGE: TTI

Incorporating encrypted mail into inbox...

   1+ 02/28 0227-EST mrose             test message  <<mumble, mumble. >>
```

Figure 5
Receiving Encrypted Mail

user directs MH to send the draft in encrypted form. The resulting output is verbose (a default for *send* for this user) but instructive. Initially, all addresses in the draft are verified on lines 14 to 17. Two forms of verification occur: first, the MTS is asked to verify the address as much as possible. For local addresses, the MTS decides if the name has a maildrop associated with it. For remote addresses, the MTS decides if the host is known to it. The second type of verification occurs with the TMA. For all addresses, the TMA is asked if it can find a mapping from the address to a KDS ID.

The reason MH goes to all this trouble is a philosophical issue. Since the copy of the encrypted draft is different for each recipient, *post* tries to verify that all recipients can be successfully posted prior to actually posting the different ciphertext versions of the draft. This behavior is not optimal in terms of cycles, but is perhaps "correct" from a UA perspective.

Finally, the draft is actually posted, and the folder carbon-copy is filed.

Some time later, the UCI portal is informed that new mail has arrived. As shown in Figure 5, the *inc* program is run. The ``E'' prior to the date of the message indicates that *inc* has detected the message to be encrypted. Since the user did not inhibit *inc* from deciphering the message, it proceeds to do so.

Finally, it may be instructive to see what the encrypted message looked like when it was delivered to the portal's maildrop, and the final message after deciphering. Figures 6 and 7 show these respectively. In particular, note that the ``X-KDS-ID:'' field has been introduced in Figure 7 after successfully deciphering the message. The presence of this field authenticates the sender of the message.

```
Received: From localhost.DELAWARE by udel-dewey.DELAWARE id a022713
          ;28 Feb 85 2:27 EST
To: uci@udel-dewey
Subject: test message
Date: 28 Feb 85 02:27:16 EST (Thu)
Message-ID: <4057.478423636@udel-dewey>
From: mrose@udel-dewey

ENCRYPTED MESSAGE: TTI TMA
(
MCL/MAIL
RCV/3
ORG/17
IDK/850228072730
KD/e36813a3882eebd1
KD/fa8b8ac657476669
IV/Ef9d283565431b103
MIC/fdb927fb
MAC/50e9de30
)
a13774f652d844762c4fc03c2f4e201b9d2f57eadb00546c
```

Figure 6
Message Prior to Decryption

```
Received: From localhost.DELAWARE by udel-dewey.DELAWARE id a022713
          ;28 Feb 85 2:27 EST
To: uci@udel-dewey
Subject: test message
Date: 28 Feb 85 02:27:16 EST (Thu)
Message-ID: <4057.478423636@udel-dewey>
From: mrose@udel-dewey
X-KDS-ID: 17 (Marshall T. Rose)

mumble, mumble.
```

Figure 7
Message After Decryption

Appendix B: A Short Exchange

The simple nature of the interchange between the user and MH in Appendix A completely hides any interactions between the TMA and the KDS. Let us briefly examine an exchange that might occur after the destination TMA receives the message shown in Figure 6.

To begin, the TMA must ascertain what it knows about the sender of the message, which claims to have a KDS ID of 17. That is, the TMA must first consider what key relationships it has with the sender. For the sake of argument,

```
 1  <--- (
 2  <--- MCL/RIU
 3  <--- RCV/17
 4  <--- ORG/3
 5  <--- KDC/TTI
 6  <--- EDC/1a1fbbba
 7  <--- )
 8  ---> (
 9  ---> MCL/RTR
10  ---> RCV/17
11  ---> ORG/3
12  ---> CTA/1
13  ---> USR/"Marshall T. Rose"
14  ---> KDC/TTI
15  ---> MAC/2ebde134
16  ---> EDC/96b183de
17  ---> )
18  <--- (
19  <--- MCL/ACK
20  <--- RCV/17
21  <--- ORG/3
22  <--- KDC/TTI
23  <--- EDC/59a8ddcc
24  <--- )
```

Figure 8
Ascertaining the Sender

suppose that this purported subscriber is unknown to the TMA. In this case, the first step it must undertake is to ascertain the validity of this subscriber.

As shown in Figure 8 on lines 1–7, the TMA does this by establishing a connection to the KDS and issuing an *request identified user* (RUI) MCL.[13] If the response by the KDS is positive, the TMA will use the information returned when generating the ``X-KDS-ID:'' field for authentication. The response CSM returned by the KDS includes an *authentication checksum* (the MAC field on line 15) and a *transaction count* (the CTA field on line 12) to prevent spoofing by a process pretending to be the KDS. The TMA then acknowledges that the response from the server was acceptable on lines 18–24.

The next step is to ascertain the actual key relationship used to encrypt the structure m, which appears after the identifying string. The TMA consults the

[13] In point of fact, the *very* first thing that the TMA does after connecting to the KDS is verify that the key relationships between the KDS and the TMA are valid (have not expired). If the key relationship between the two has expired, the TMA issues a *request service initialization* RSI MCL to establish a new key relationship. This relationship contains a *key-encrypting key* (KK) and an *authentication key* (KA). Once a valid key relationship exists between the KDS and the TMA, transactions concerning other key relationships may take place.

```
 1  <--- (
 2  <--- MCL/RSI
 3  <--- RCV/17
 4  <--- ORG/3
 5  <--- IDK/850228072730
 6  <--- KDC/TTI
 7  <--- SVR/KD.IV.KK
 8  <--- EDC/83679e14
 9  <--- )
10  ---> (
11  ---> MCL/RTR
12  ---> RCV/17
13  ---> ORG/3
14  ---> KK/095f9d6b87f57871
15  ---> CTA/2
16  ---> KD/527fbb5593efd318
17  ---> KD/1dcab338be1e7a09
18  ---> IV/E02db5e598b2823ae
19  ---> EDK/850618075332
20  ---> KDC/TTI
21  ---> MAC/12cbbdf5
22  ---> EDC/8cd0c4a8
23  ---> )
24  <--- (
25  <--- MCL/ACK
26  <--- RCV/17
27  <--- ORG/3
28  <--- KDC/TTI
29  <--- EDC/59a8ddcc
30  <--- )
```

Figure 9
Ascertaining the Key Relationship

IDK field in m, and if this relationship is unknown to it, then the KDS is asked to disclose the key relationship.

As shown in Figure 9 on lines 1–9, This is done by issuing a *request service initialization* (RSI) MCL and specifying the particular key relationship of interest. The KDS consults its database, and if the exact key relationship between the two indicated TMAs can be ascertained, it returns this information. The key relationship is encrypted using the key relationship between the KDS and the TMA, and the usual count and authentication fields are included.

Once the TMA knows the key relationship used to encrypt the structure m, it can decider the structure and ascertain the KD/IV/KA triple used to encrypt the body of the message.

Appendix C: Differences between the ANSI and TTI drafts

The differences between the ANSI draft standard for financial institution key management, and the *TTI* draft's specification for trusted mail handling, are considered.

The concept of a *key distribution center* (CKD in the ANSI draft, KDC in the *TTI* draft) environment differs. In the ANSI draft, only one party talks to the *key distribution server* (KDS); in the *TTI* draft, both parties talk to the KDS. This leads to a number of differences in the two protocols. The reason for this shift in the *TTI* draft is somewhat subtle: although both parties can talk to the KDS, the *mail transfer system* (MTS) environment is such that both *user agents* (UAs) are unable to contact each other in real-time. Hence, a detailed two-way protocol between them is prohibitively expensive.[14]

Before discussing the differences between the two drafts, let us consider the differences in the two environments: in the electronic mail environment, the two end-to-end peers need not be simultaneously online. Electronic mail relies on a communication service with potentially large delays in transit between *message transfer agents* (MTAs). A basic concept of "mail" is that an originator must release the enveloped message to a "transfer agent" before delivery can be attempted to a recipient. In contrast, in the electronic funds environment, the two peers make use of a virtual-circuit service. This means that they can synchronize much easier and inter-operate in a more direct fashion.

Service protocols are based on the notion of requests and responses. A client issues a request to a server, the server processes the request and returns a response. Depending on the complexity of the protocol, the client may now respond to the server's message, or might issue a new request, or might terminate the connection.

As delays in the network increase, along with the possibility of loss or corruption or re-ordering of messages, it becomes more difficult to implement a service protocol. In the case of a high-level protocol making use of a virtual-circuit service, most problems can be ignored, as the virtual-circuit service masks out problems in the network by using sequences, positive (and/or negative) acknowledgments, windows, and so on.

Sadly, electronic mail cannot utilize a virtual-circuit throughout the MTS (although individual MTA-wise connections are (in theory) virtual-circuit based). This means that implementing a real-time or interactive service protocol between two endpoints (a.k.a. UAs) in the MTS is very difficult. As a result, the complexity of an end-to-end protocol in the MTS (in terms of requests and responses) is severely constrained. For all practical purposes, an MTA can assume datagram service and nothing else: messages might be re-ordered; messages might not reach

[14] In the words of Einar A. Stefferud: "Every interesting connection has at least two end-points — connections with only one end-point are always uninteresting."

their destination; messages might be corrupted (though this is unlikely); in cases of failure, a notice might be generated, or might not.

In terms of the environment in which *cryptographic service messages* (CSMs) must flow, the high degree of delay and uncertainty make the implementation of a complex end-to-end protocol between UAs prohibitively expensive. Hence, a KDC is needed, to which each UA can connect using a virtual-circuit service, at posting and delivery time. The *TTI* draft terms such a user agent a *trusted mail agent* (TMA). Since both TMAs can connect to the KDS at different times using different media, the KDS maintains state information about the key relationships between different TMAs and manages those relationships appropriately. Since connections to the KDS can be expensive in terms of resources, each TMA caches information received from the KDS appropriately.

That's the gist of the argument as to why the *TTI* draft differs from the ANSI draft. It might be possible to include CSMs in the messages which UAs exchange, but management of these CSMs can not be done reliably or in a straightforward fashion owing to the datagram nature of the service offered by the MTS. Finally, it should be noted that in the *TTI* draft, the KDS never initiates a connection with a TMA, rather it is the TMAs which connect to the KDS.

In the following, the differences between the two drafts are highlighted. Minor differences between the two are not discussed.

In the ANSI draft, §4.2 (p. 22) discusses the requirements for the automated key management architecture. The *TTI* draft has somewhat more "depth", since the ANSI draft does not make use of a *master key* (MK) to fully automate the distribution of *key-encrypting keys* (KK).

The ANSI draft states that once a KK-relationship is discontinued by either of that pair, the relation is not to be re-used for any subsequent activity. This can't be guaranteed in the prototype implementation. If one of the TMAs wishes to discontinue a key, not only does it have to inform the KDS, but the other TMA as well. Since the *TTI* draft does not permit CSMs between TMA-peers, the latter action doesn't seem possible. However, there is a solution. Whenever a message is deciphered, the TMA checks the effective date of the key used to encrypt a message it has received, and if the key is newer than the one it currently uses, it considers the older key to be discontinued.

Furthermore, although the environment in the *TTI* draft is that of a key distribution center, the notion of an *ultimate recipient* is not present, since all clients connect to the KDS at one time or another. In addition, the differences between the environs envisioned by the two drafts become even more pronounced when one considers that the KDS distributes key-encrypting keys to TMAs, although the ANSI draft specifically prohibits this.

Finally, there is another important technical difference between the two drafts: every request to the KDS by the TMA results in a specifically defined response from the KDS to the TMA. Furthermore, if the KDS responds in a positive manner, then the TMA acknowledges this. This three-way interaction is used to ensure consistency between the states of the KDS and the TMA. The ANSI draft does not require such behavior, and might profit from some finite-state analysis to ascertain unsafe (in terms of correctness) states which are reachable.

Contents

This document (version #2.60) was TEXset April 12, 1990 with DISS.STY v103.