# Limitations of the Kerberos Authentication System†

Steven M. Bellovin − AT&T Bell Laboratories
Michael Merritt − AT&T Bell Laboratories

## ABSTRACT

The Kerberos authentication system, a part of MIT's Project Athena, has been adopted by other organizations. Despite Kerberos's many strengths, it has a number of limitations and some weaknesses. Some are due to specifics of the MIT environment; others represent deficiencies in the protocol design. We discuss a number of such problems, and present solutions to some of them. We also demonstrate how special-purpose cryptographic hardware may be needed in some cases.

## INTRODUCTION

The Kerberos authentication system[Stei88, Mill87, Brya88] was introduced by MIT to meet the needs of Project Athena. It has since been adopted by a number of other organizations for their own purposes, and is being discussed as a possible standard. In our view, both these decisions may be premature. Kerberos has a number of limitations and weaknesses; a decision to adopt or reject it cannot properly be made without considering these issues. (A *limitation* is a feature that is not as general as one might like, while a *weakness* could be exploited by an attacker to defeat the authentication mechanism.) Some improvements can be made within the current design. Support for optional mechanisms would extend Kerberos's applicability to environments radically different from MIT.

These problems fall into several categories. Some stem from the Project Athena environment. Kerberos was designed for that environment; if the basic assumptions differ, the authentication system may need to be changed as well. Other problems are simply deficiencies in the protocol design. Some of these are corrected in the proposed Version 5 of Kerberos,[Kohl89] but not all. Even the solved problems merit discussion, since the code for Version 4 has been widely disseminated. Finally, some problems with Kerberos are not solvable without employing special-purpose hardware, no matter what the design of the protocol. We will consider each of these areas in turn.

We wish to stress that we are not suggesting that Kerberos is useless. Quite the contrary — an attacker capable of carrying out any of the attacks listed here could penetrate a typical network of UNIX systems far more easily. Adding Kerberos to a network will, under virtually all circumstances, significantly increase its security; our criticisms focus on the extent to which security is improved. Further, we recommend changes to the protocols that substantially increase security.

Beyond its specific utility in production, Kerberos serves a major function by focusing interest on practical solutions to the network authentication problem. The elegant protocol design and wide availability of the code has galvanized a wide audience. Far from a condemnation, our critique is intended to contribute to an understanding of Kerberos's properties and to influence its evolution into a tool of greater power and utility.

Several of the problems we point out are mentioned in the original Kerberos paper or elsewhere.[Davi90] For some of these, we present protocol improvements that solve, or at least ameliorate, the problem; for others, we place them squarely in the context of the intended Kerberos environment.

### Version 5, Draft 3

Since this paper was written, a new draft of the Version 5 protocol has been released, and a final specification is promised.[Kohl90] Many of the problems we discuss herein have been corrected. Others remain, and we have found a few new ones. The ultimate resolution of these issues is unclear as we go to press. Consequently, a brief analysis of Draft 3 is presented in an appendix, rather than in the main body of the document.

### Focus on Security

Kerberos is a security system; thus, though we address issues of functionality and efficiency, our primary emphasis is on the security of Kerberos in a general environment. This means that security-critical assumptions must be few in number and stated clearly. For the widest utility, the network must be considered as completely open. Specifically, the protocols should be secure even if the network is

†A version of this paper was published in the October, 1990 issue of *Computer Communications Review*.

under the complete control of an adversary.[1] This means that defeating the protocol should require the adversary to invert the encryption algorithm or to subvert a principal specifically assumed to be trustworthy. Only such a strong design goal can justify the expense of encryption. (No ''steel doors in paper walls''.) We believe that Kerberos can meet this ambitious goal with only minor modifications, retaining its essential character.

Some of our suggestions bear a performance penalty; others complicate the design of suggested enhancements. As more organizations make use of Kerberos, pressures to enhance or augment its functionality and efficiency will increase. Security has real costs, and the benefits are intangible. There must be a continuing and explicit emphasis on security as the overriding requirement.

### Validation

It is not sufficient to design and implement a security system. Such systems, though apparently adequate when designed, may have serious flaws. Consequently, systems must be subjected to the strongest scrutiny possible. A consequence of this is that they must be designed and implemented in a manner that facilitates such scrutiny. Kerberos has a number of problems in this area as well.

## WHAT'S A KERBEROS?

Before discussing specific problem areas, it is helpful to review Kerberos Version 4. Kerberos is an *authentication* system; it provides evidence of a *principal*'s identity. A principal is generally either a user or a particular service on some machine. A principal consists of the three-tuple

$$< primaryname,\ instance,\ realm >.$$

If the principal is a user — a genuine person — the *primary name* is the login identifier, and the *instance* is either null or represents particular attributes of the user, i.e., `root`. For a service, the service name is used as the primary name and the machine name is used as the instance, i.e., `rlogin.myhost`. The *realm* is used to distinguish among different authentication domains; thus, there need not be one giant — and universally trusted — Kerberos database serving an entire company.

[1]The Project Athena Technical Plan[Mill87, section 2] describes a simpler threat environment, where eavesdropping and host impersonation are of primary concern. While this may be appropriate for MIT, it is by no means generally true. Consider, for example, a situation where general-purpose hosts also function as routers, and packet modification or deletion become significant concerns.

Table 1: Notation

| | |
|---|---|
| $c$ | client principal |
| $s$ | server principal |
| $tgs$ | ticket-granting server |
| $K_x$ | private key of ''$x$'' |
| $K_{c,s}$ | session key for ''$c$'' and ''$s$'' |
| $\{info\}K_x$ | ''$info$'' encrypted in key $K_x$ |
| $\{T_{c,s}\}K_s$ | Encrypted ticket for ''$c$'' to use ''$s$'' |
| $\{A_c\}K_{c,s}$ | Encrypted authenticator for ''$c$'' to use ''$s$'' |
| $addr$ | client's IP address |

Kerberos principals may obtain *tickets* for services from a special server known as the *ticket-granting server*, or *TGS*. A ticket contains assorted information identifying the principal, encrypted in the private key of the service. (Notation is summarized in Table 1.)

$$\{T_{c,s}\}K_s = \{s,\ c,\ addr,\ timestamp,\ lifetime,\ K_{c,s}\}K_s$$

Since only Kerberos and the service share the private key $K_s$, the ticket is known to be authentic. The ticket contains a new private session key, $K_{c,s}$, known to the client as well; this key may be used to encrypt transactions during the session.[2]

To guard against *replay attacks*, all tickets presented are accompanied by an *authenticator*:

$$\{A_c\}K_{c,s} = \{c,\ addr,\ timestamp\}K_{c,s}$$

This is a brief string encrypted in the session key and containing a timestamp; if the time does not match the current time within the (predetermined) clock skew limits, the request is assumed to be fraudulent.

For services where the client needs bidirectional authentication, the server can reply with

$$\{timestamp + 1\}K_{c,s}$$

This demonstrates that the server was able to read *timestamp* from the authenticator, and hence that it knew $K_{c,s}$; that in turn is only available in the ticket, which is encrypted in the server's private key.

Tickets are obtained from the TGS by sending a *request*

$$s,\ \{T_{c,tgs}\}K_{tgs},\ \{A_c\}K_{c,tgs}$$

In other words, an ordinary ticket/authenticator pair is used; the ticket is known as the *ticket-granting*

[2]Technically speaking, $K_{c,s}$ is a *multi-session key*, since it is used for all contacts with that server during the life of the ticket.

*ticket.* The TGS responds with a ticket for server *s* and a copy of $K_{c,s}$, all encrypted with a private key shared by the TGS and the principal:

$$\{\{T_{c,s}\}K_s, K_{c,s}\}K_{c,tgs}$$

The session key $K_{c,s}$ is a newly-chosen random key.

The key $K_{c,tgs}$ and the ticket-granting ticket itself, are obtained at session-start time. The client sends a message to Kerberos with a principal name; Kerberos responds with

$$\{K_{c,tgs}, \{T_{c,tgs}\}K_{tgs}\}K_c$$

The client key $K_c$ is derived from a non-invertible transform of the user's typed password. Thus, all privileges depend ultimately on this one key.

Note that servers must possess private keys of their own, in order to decrypt tickets. These keys are stored in a secure location on the server's machine.

## THE KERBEROS ENVIRONMENT

The Project Athena computing environment consists of a large number of more or less anonymous workstations, and a smaller number of large autonomous server machines. The servers provide volatile file storage, print spooling, mailboxes, and perhaps some computing power; the workstations are used for most interaction and computing. Generally, they possess local disks, but these disks are effectively read-only; they contain no long-term user data. Furthermore, they are not physically secure; someone so inclined could remove, read, or alter any portion of the disk without hindrance.

Within this environment the primary need is for user-to-server authentication. That is, when a user sits down at a workstation, that person needs access to private files residing on a server. The workstation itself has no such files, and hence has no need to contact the server or even to identify itself.

This is in marked contrast to a typical UNIX system's view of the world. Such systems do have an identity, and they do own files. Assorted network daemons transfer files in the background, clock daemons perform management functions, electronic mail and news arrives, etc. If such a machine relied on servers to store its files, it would have to assert, and possibly prove, an identity when talking to these servers. The Project Athena workstations are neither capable nor in need of such; they in effect function as very smart terminals with substantial local computing power, rather than as full computer systems.[3]

What does this mean for Kerberos? Simply this: Kerberos is designed to authenticate the end-user — the human being sitting at the keyboard — to some number of servers. It is not a peer-to-peer system; it is not intended to be used by one computer's daemons when contacting another computer. Attempting to use Kerberos in such a mode can cause trouble.[4]

We make this statement for several reasons. First and foremost, typical computer systems do not have a secure key storage area. In Kerberos, a plaintext key must be used in the initial dialog to obtain a ticket-granting ticket. But storing plaintext keys in a machine is generally felt to be a bad idea;[Morr79] if a Kerberos key that a machine uses for itself is compromised, the intruder can likely impersonate any user on that computer, by impersonating requests vouched for by that machine (i.e., file mounts or `cron` jobs).[5] Additionally, the session keys returned by the TGS cannot be stored securely; of necessity, they are stored in some area accessible to `root`. Thus, if the intruder can crack the protection mechanism on the local computer — or, perhaps more to the point, work around it for some limited purposes — all current session keys can be stolen. This is less serious than a breach of the primary Kerberos key, of course, since session keys are limited in lifetime and scope; nevertheless, one does not wish these keys exposed.

This points out a second flaw when multi-user computers employ Kerberos, either on their own behalf or for their users: the cached keys are accessible to attackers logged in at the same time. In a workstation environment, only the current user has access to system resources; there is little or no need even to enable remote login to that workstation. There are many reasons for this; a consequence, though, is that the intruder simply cannot approach the safe door to try to pick its lock.[6] Only when the legitimate user leaves can the attacker attempt to find the keys. But the keys are no longer available; Kerberos attempts to wipe out old keys at logoff time, leaving the attacker to sift through the debris. With a multi-user computer, on the other hand, an attacker has concurrent access to the keys if there are flaws in the host's security.

There are two other minor flaws in Kerberos directly attributable to the environment. First, there is some question about where keys should be cached. Since all of the Project Athena machines have local disks, the original code used `/tmp`. But this is highly insecure on diskless workstations, where `/tmp` exists on a file server; accordingly, a modification was made to store keys in shared memory. However, there is no guarantee that shared memory is not paged; if this entails network traffic,

[3] We regard this as a feature, not a bug.

[4] More precisely, Kerberos is not a *host-to-host* protocol. In Version 5, it has been extended to support user-to-user authentication.[Davi90]

[5] Recall that we are assuming here that the machine — and hence its superuser — needs an identity of its own.

[6] On Project Athena machines, remote access to most workstations is in fact disabled.

an intruder can capture these keys.

Finally, the Kerberos protocol binds tickets to IP addresses. Such usage is problematic on on multi-homed hosts (i.e., hosts with more than one IP address). Since workstations rarely have multiple addresses, this feature — intended to enhance security — was not a problem at MIT. Multi-user hosts often do have multiple addresses, however, and cannot live with this limitation. This problem has been fixed in Version 5.

## PROTOCOL WEAKNESSES

### Replay Attacks

The Kerberos protocol is not as resistant to penetration as it should be. A number of weaknesses are apparent; the most serious is its use of an authenticator to prevent replay attacks.

The authenticator relies on use of a timestamp to guard against reuse. This is problematic for several reasons. The claim is made that no replays are likely within the lifetime of the authenticator (typically five minutes). This is reinforced by the presence of the IP address in both the ticket and the authenticator. We are not persuaded by this logic. An intruder would not start by capturing a ticket and authenticator, and then develop the software to use them; rather, everything would be in place before the ticket-capture was attempted. Let us consider two examples.

Some years ago, Morris described an attack based on the slow increment rate of the initial sequence number counter in some TCP implementations.[Morr85] He demonstrated that it was possible, under certain circumstances, to spoof one half of a preauthenticated TCP connection without ever seeing any responses from the targeted host. In a Kerberos environment, his attack would still work if accompanied by a stolen live authenticator, but not if a challenge/response protocol was used. Alternatively, an intruder may simply watch for a ''mail-checking'' session, wherein a user logs in briefly, reads a few messages, and logs out. A number of valuable tickets would be exposed by such a session, notably the one used to mount the user's home directory. Note that the lifetime of the authenticators — 5 minutes — contributes considerably to this attack.

Further, the proposed Version 5 of Kerberos anticipates alternative communication protocols in which such replays may be trivial to implement. If Kerberos is to be considered as a general-purpose utility, it must make few security-critical assumptions about the underlying network, and those must be explicit.

It has been suggested that the proper defense is for the server to store all live authenticators; thus, an attempt to reuse one can be detected.[Stei88] In fact, the original design of Kerberos required such caching, though this was never implemented. (While that is a feature of the implementation rather than of the protocol itself, a security feature is not very useful if it is too hard to implement.)

For several reasons, we do not think that caching solves the problem. First, on UNIX systems it is difficult for TCP-based[Post81] servers to store authenticators. Servers generally operate by forking a separate process to handle each incoming request. The child processes do not share any memory with the parent process, and thus have no convenient way to inform it — and hence any other child servers — of the value of the authenticator used. There are a number of obvious solutions — pipes, authenticator servers, shared memory segments and the like — but all are awkward, and some even raise authentication questions of their own. To date, we know of no multi-threaded server implementation which caches authenticators.

UDP-based[Post80] query servers can store the authenticators more easily, as a single process generally handles all incoming requests; however, they might have problems with legitimate retransmissions of the client's request if the answer was lost. (UDP does not provide guaranteed delivery; thus, all retransmissions happen from application level, and are visible to the application.) Legitimate requests could be rejected, and a security alarm raised inappropriately. One possible solution would be for the application to generate a new authenticator when retransmitting a request; were it not for the other weaknesses of the authenticator scheme, this would be acceptable.

### Secure Time Services

As noted, authenticators rely on machines' clocks being roughly synchronized. If a host can be misled about the correct time, a stale authenticator can be replayed without any trouble at all. Since some time synchronization protocols are unauthenticated,[Post83, Mill88] and hosts are still using these protocols despite the existence of better ones,[Mill89] such attacks are not difficult.

The design philosophy of building an authentication service on top of a secure time service is itself questionable. That is, it may not make sense to build an authentication system assuming an already-authenticated underlying system. Furthermore, while spoofing an unauthenticated time service may be a difficult programming task, it is not cryptographically difficult.[7] Using time-based protocols in a secure fashion means thinking through all these issues carefully and making the appropriate

[7] In some environments, programming is not even necessary. Low-powered fake WWV transmitters are not hard to build, and, if properly located, could easily block out the legitimate signal.

synchronization an explicit part of the protocol. As Kerberos is proposed for more varied environments, its dependence on a secure time service becomes more problematic and must be stressed.

As an alternative, we propose the use of a challenge/response authentication mechanism. As is done today, the client would present a ticket, though without an authenticator. The server would respond with a nonce identifier encrypted with the session key $K_{c,s}$; the client would respond with some function of that identifier, thereby proving that it possesses the session key.

Such an implementation is not without its costs, of course. An extra pair of messages must be exchanged each time a ticket is used, which rules out the possibility of authenticated datagrams. More seriously, all servers must then retain state to complete the authentication process. While not a problem for TCP-based servers, this may require substantial modification to UDP-based query servers. (The complexity of managing outstanding challenges may be comparable to that needed to cache live authenticators — the trade-off is not between a stateful and a stateless protocol, but in managing two kinds of state.)

There is a signficant philosophical difference between the two techniques, however. In the current Kerberos implementation, with its assumptions about the network environment, retained state is only necessary to enhance security. The challenge/response scheme, on the other hand, guarantees security in a more general environment, but requires retained state to function at all.

Instead of substituting challenge/response throughout, a possible compromise is to extend the protocol with a challenge/response option. This option could be used, for example, to authenticate the user in the initial ticket-granting ticket exchange and to access a time service.[8] Subsequent client-server interactions could use the current time-based protocol. But synchronizing the servers remains a problem; not synchronizing them will lead to denial of service, and if they access the time service as a client, they must somehow obtain and store a ticket and key to authenticate it. (See above on storing keys in servers.) Given these complexities and possible weaknesses, it would seem reasonable to allow any service to insist on the challenge/response option.

Summarizing, we emphasize that the security of Kerberos depends critically on synchronized clocks. In essence, the Kerberos protocols involve mutual trust among four parties: the client, server, authentication server and time server.

[8]This was suggested to us by Clifford Neuman.

**Password-Guessing Attacks**

A second major class of attack on the Kerberos protocols involves an intruder recording login dialogs in order to mount a password-guessing assault. When a user requests $T_{c,tgs}$ (the ticket-granting ticket), the answer is returned encrypted with $K_c$, a key derived by a publicly-known algorithm from the user's password. A guess at the user's password can be confirmed by calculating $K_c$ and using it to decrypt the recorded answer. An intruder who has recorded many such login dialogs has good odds of finding several new passwords; empirically, users do not pick good passwords unless forced to.[Morr79, Gram84, Stol88]

We propose the use of exponential key exchange[Diff76] to provide an additional layer of encryption. Without describing the algorithm in detail, it involves the two parties exchanging numbers that each can use to compute a secret key. An outsider, not knowing how the numbers were calculated, cannot easily derive the key.

Such a use of exponential key exchange would prevent a passive wiretapper from accumulating the network equivalent of `/etc/passwd`. While exponential key exchange is normally vulnerable to active wiretaps, such attacks are comparatively rare, especially if dedicated network routers are used.

Apart from licensing issues — exponential key exchange is protected by a U.S. patent — using it has its costs. LaMacchia and Odlyzko[LaMa] have demonstrated that exchanging small numbers is quite insecure, while using large ones is expensive in computation time. Additionally, we have added extra messages to the login dialog, and imposed the requirement for considerable extra state in the server. Given the trend towards hiding even encrypted passwords on UNIX systems, and given estimates that half of all logins at MIT are used within a two-week period, the investment may be justifiable. Perhaps the best solution is to support this feature as a domain-specific option.

Even exponential key exchange will not prevent all password-guessing attacks. Depending on how carefully the Kerberos logs are analyzed, an intruder need not even eavesdrop. Requests for tickets are not themselves encrypted; an attacker could simply request ticket-granting tickets for many different users. An enhancement to the server, to limit the rate of requests from a single source, may be useful.

Alternatively, some portion of the initial ticket request may be encrypted with $K_c$, providing a minimal authentication of the user to Kerberos, such that true eavesdropping would be required to mount this attack. (As we are preparing this manuscript, just such a suggestion is being hotly debated on the Kerberos mailing list. We originally overlooked an alternative avenue for mounting a password-guessing attack. Clients may be treated as services, and

tickets to the client, encrypted by $K_c$, may be obtained by any user. This capability has been suggested as the basis for user-to-user authentication and and enhanced mail services.[Salt90] But any such scheme would seem to require repeated re-entry of the user's password, an inconvenience we suspect will not be tolerated. We would prefer to provide the same functionality by having clients register separate instances as services, with truly random keys. Keys could be supplied to the client by the *keystore*, described below.)

An alternative approach is a protocol described by Lomas, Gong, Saltzer, and Needham.[Loma89] They present a dialog with a server that does not expose the user to password-guessing attacks. However, their protocol relies on public-key cryptography, an approach explicitly rejected for Kerberos.

### Spoofing Login

In a workstation environment, it is quite simple for an intruder to replace the `login` command with a version that records users' passwords before employing them in the Kerberos dialog. Such an attack negates one of Kerberos's primary advantages, that passwords are never transmitted in cleartext over a network. While this problem is not restricted to Kerberos environments, the Kerberos protocol makes it difficult to employ the standard countermeasure: one-time passwords.

A typical one-time password scheme employs a secret key shared between a server and some device in the user's possession. The server picks a random number and transmits it to the user. Both the server and the user (with the aid of the device) encrypt this number using the secret key; the result is transmitted back to the server. If the two computed values match, the user is assumed to possess the appropriate key.

Kerberos makes no provision for such a challenge/response dialog at login time. The server's response to the login request is always encrypted with $K_c$, a key derived from the user's password. Unless a ''smart card'' is employed that understands the entire Kerberos protocol, this precludes any use of one-time passwords.

An alternative (first suggested to us by T.H. Foregger) requires that the server pick a random number $R$, and use $K_c$ to encrypt $R$. This value $\{R\}K_c$, rather than $K_c$, would be used to encrypt the server's response. $R$ would be transmitted in the clear to the user. If a hand-held authenticator was in use, the user would employ it to calculate $\{R\}K_c$; otherwise, the login program would do it automatically.

Several objections may be raised to this scheme. First, hand-held authenticators are often thought to be inconvenient. This is true; however, they offer a substantial increase in security in high-threat environments. If they are not used, the cost of our scheme is quite low, simply one extra encryption on each end.

A second, more cogent, objection is that if the client's workstation cannot be trusted with a user's password, it cannot be trusted with session keys provided by Kerberos. This is, to some extent, a valid criticism, though we believe that compromise of the login password is much more serious than the capture of a few limited-lifetime session keys. This problem cannot be solved without the use of special-purpose hardware, a subject we shall return to below.

Finally, it has been pointed out that a user can always supply a known-clean boot device, or boot via the network. The former we regard as improbable in practice unless removable media are employed; the latter is insecure because the boot protocols are unauthenticated.

### Inter-Session Chosen Plaintext Attacks

According to the description in the Version 5 draft,[Kohl89] servers using the KRB_PRIV format are susceptible to a *chosen plaintext attack*. (A chosen-plaintext attack is one where an attacker may choose all or part of the plaintext and, typically, use the resulting cipher text to attack the cipher. Here we use the cipher text to attack the protocol. Mail and file servers are examples of servers susceptible to such attacks.) Specifically, the encrypted portion of messages of this type have the form

$$X = (DATA,\ timestamp + direction,\ hostaddress,\ PAD)$$

Since cipher-block chaining[FIPS81, Davi89] has the property that prefixes of encryptions are encryptions of prefixes, if *DATA* has the form

$$(AUTHENTICATOR,\ CHECKSUM,\ REMAINDER)$$

then a prefix of the encryption of $X$ with the session key is the encryption of

$$(AUTHENTICATOR,\ CHECKSUM),$$

and can be used to spoof an entire session with the server.

It may be argued that most servers are not susceptible to chosen plaintext attacks. Given that there are easy counters to this attack, it seems foolish to advocate a general format for private servers that does not also protect against it.

It should be noted that the simple attack above does not work against Kerberos Version 4, in which the encrypted portion of the KRB_PRIV message is of the form

$$(length(DATA),\ DATA,\ msectime,\ hostaddress,\ timestamp + direction,\ PAD)$$

as the leading $length(DATA)$ field disrupts the prefix-based attack. We leave it to the reader to

discover a more complicated chosen ciphertext attack against this format, even allowing for the fact that Version 4 uses the nonstandard PCBC mode of encryption. (Hint: assume the initial vector is fixed and public.) However, it is interesting to note that the order of concatenation of message fields can have security-critical implications. We return to this question in the later section on message encoding.

**Exposure of Session Keys**

The term ''session key'' is a misnomer in the Kerberos protocol. This key is contained in the service ticket and is used in the multiple sessions between the client and server that use that ticket. Thus, it is more properly called a ''multi-session key''. Making this point explicit leads naturally to the suggestion that true session keys be negotiated as part of the Kerberos protocol. This limits the exposure to cryptanalysis[Kahn67, Beke82, Deav85] of the multi-session key contained in the ticket, and precludes attacks which substitute messages from one session in another. (The chosen-plaintext attack of the previous section is one such example.) The session key could be generated by the server or could be computed as a session-specific function of the multi-session key.

**The Scope of Tickets**

Kerberos tickets are limited in both time and space. That is, tickets are usable only within the realm of the ticket-granting server, and only for a limited period of time. The first is necessary to the design of Kerberos; the TGS would not have any keys in common with servers in other realms. The latter is a security measure; the longer a ticket is in use, the greater the risk of it being stolen or compromised.

A further restriction on tickets, in Version 4, is that they cannot be forwarded. A user may obtain tickets at login time, and use these to log in to some other host; however, it is not possible to obtain authenticated network services from that host unless a new ticket-granting ticket is obtained. And that in turn would require transmission of a password across the network, in violation of fundamental principles of Kerberos's design.[9]

Version 5 incorporates provisions for ticket-forwarding; however, this introduces the problem of cascading trust. That is, a host *A* may be willing to trust credentials from host *B*, and *B* may be willing to trust host *C*, but *A* may not be willing to accept tickets originally created on host *C*, which *A* believes to be insecure. Kerberos has a flag bit to indicate that a ticket was forwarded, but does not include the original source.

A second problem with forwarding is that the concept only makes sense if tickets include the network address of the principal. If the address is omitted — as is permitted in Version 5 — a ticket may be used from any host, without any further modifications to the protocol. All that is necessary to employ such a ticket is a secure mechanism for copying the multi-session key to the new host. But that can be accomplished by an encrypted file transfer mechanism layered on top of existing facilites; it does not require flag bits in the Kerberos header.

Is it useful to include the network address in a ticket? We think not. Given our assumption that the network is under full control of the attacker, no extra security is gained by relying on the network address. In fact, the primary benefit of including it appears to be preventing immediate reuse of authenticators from a different host.

Even with the protection provided by network addresses, replay attacks that involve faked addresses are easy; again, see [Morr85]. Furthermore, an attacker can always wait until the connection is set up and authenticated, and then take it over, thus obviating any security provided by the presence of the address. Given these problems, and the cascading trust issue raised earlier, we suggest that ticket-forwarding be deleted.

A new inter-realm authentication mechanism is also introduced in Version 5. Briefly, if a user wishes to access a service in another realm, that user must first obtain a ticket-granting ticket for that realm. This is done by making the ticket-granting server in a realm the client of another realm's TGS. It in turn may be a client of yet another realm's TGS. A user's ticket request is signed by each TGS and passed along; realms will normally be configured in a hierarchical fashion, though ''tandem links'' are permitted.

Unfortunately, this scheme, while appearing to solve the problem, is deficient in several respects. First, and most serious, there is no discussion of how a TGS can determine which of its neighboring realms should be the next hop. Moving up the tree, towards the root, is an obvious answer for leaf nodes; however, each parent node would need complete knowledge of its entire subtree's realms in order to determine how to pass the request downwards. There are obvious analogies here to network-layer routing issues; note, though, that any ''realm routing protocol'' must include strong authentication provisions.

Another answer is to say that static tables should be used. This, too, has its security limitations: should realm administrators rely on electronic mail messages or telephone calls to set up their

[9] Actually, a special-purpose ticket-forwarder was built for Version 4. However, the implementation was of necessity awkward, and required participating hosts to run an additional server.

routing tables? If such calls are not authenticated, the security risks are obvious; if they are, the security of a Kerberos realm is subordinated to the security of a totally different authentication system.

There is also an evident link between inter-realm authentication and the cascading-trust problem. Kerberos Version 5 attempts to solve this by including path information in the ticket request. However, in the absence of a global name space, it is not clear that this is useful. If a realm is not a neighbor, its name may not carry any global signficance, whether by malice or coincidence. Furthermore, to assess the validity of a request, a server needs global knowledge of the trustworthiness of all possible transit realms. In a large internet, such knowledge is probably not possible.

## KERBEROS HARDWARE DESIGN CRITERIA

### A Host Encryption Unit

One of the major reasons we question the suitability of Kerberos for multi-user hosts is the need for plaintext key storage. What if the host were equipped with an attached cryptographic unit? We consider the design parameters for such a box.

The primary goal is to perform cryptographic operations without exposing any keys to compromise. These operations must include validating tickets presented by remote users, creating requests for both ticket-granting tickets and application tickets, and encrypting and decrypting conversations. Consequently, there must be secure storage for an adequate number of keys, and the operating system must be able to select which key should be used for which function.

The next question, of course, is how keys are entered into the secure storage area. If tickets are decrypted by the encryption box but transferred to the host's memory for analysis, the embedded session key is exposed.[10] Therefore, we conclude that the encryption box itself must understand the Kerberos protocols; nothing less will guarantee the security of the stored keys.

Entry of user keys is more problematic, since they must travel through the host. Unless user terminals are connected directly to the encryption unit, there is little choice. Storing them off the host, though, is a significant help, as the period of exposure is then minimized. Host-owned keys — service keys, or the keys that `root` would use to do NFS mounts — should be loaded via a Kerberos-authenticated service resident in the encryption unit. We shall return to this point below.

We must now ensure that the protocol itself does not provide a mechanism to obtain keys. Looking at the message definitions, we see that only session keys are ever sent, and these are always sent encrypted. Furthermore, user machines never generate any such messages; they merely forward them. Thus, the box need not have the ability to transmit a key, thereby providing us with a very high level of assurance that it will not do so.

If an encryption box is used for the Kerberos server itself, the problem is somewhat more complex. There are two places where keys are transmitted. First, when a ticket is granted, the ticket itself contains a session key, and a copy of that session key is sent back encrypted in the client's ticket-granting session key. Second, during the initial dialog with Kerberos, the ticket-granting session key must be sent out, encrypted in the client's password key. Note, though, that permanent keys are *never* sent; again, this assures us that the encryption box will not give away keys. Furthermore, since these session keys are intended to be random, we can buy ourselves a great deal of security by including a hardware random number generator on-board.

We are not too concerned about having to load client and server keys onto the board. This operation is done only by the Kerberos master server, for which strong physical security must be assumed in any event. It is possible that such an encryption unit can be made sufficiently tamper-resistant that even workstations can use them; certainly, there are commercial cryptographic devices that claim such strengths.

One major objection to this entire scheme is that ultimately, the encryption box is controlled by the host computer. Thus, if `root` is compromised, the host could instruct the box to create bogus tickets. Such concerns are certainly valid. However, as noted above, we consider such temporary breaches of security to be far less serious than the compromise of a key. Furthermore, using a separate unit allows us to create untamperable logs, etc.

It is also desirable to prevent misuse of keys. For example, we do not want the login key used to decrypt the arbitrary block of text that just happens to be the ticket-granting ticket. Accordingly, keys should be tagged with their purpose. A login key should be used only to decrypt the ticket-granting ticket; the key associated with it should be used only for obtaining service tickets, etc. Since the encryption box is performing all of the key management, this is not a difficult problem.

### The Key Storage Unit

[10] This is not a hypothetical concern. A program to do just that (for conventional passwords) was posted to *netnews* as long ago as 1984. It operated by reading `/dev/kmem`. The existence of this program was a principal factor motivating the current restrictive permission settings on `/dev/kmem`.

A variety of technologies may be used to implement encryption units, ranging from special boards to dedicated microcomputers connected to server hosts by physically-secure lines. If the latter is used, there is the temptation to use its disk storage to hold the service keys associated with the attached host, but we feel that that is inadvisable. Any media of that sort must be backed up, and the backups must be carefully guarded. Such a high degree of security may be impractical in some environments. Instead, we suggest that keys be kept in volatile memory, and downloaded from a secure *keystore* on request, via an encryption-protected channel. Thus, only one master key need be stored within the box; this key could either be in non-volatile storage, or be supplied by an operator when necessary.

More generally, the keystore is a secure, reliable repository for a limited amount of information. A client of the keystore could package arbitrary data to be retained by the keystore, and retrieved at a later date. This data — the service keys and tags, in the case of an encryption unit, or even a conventional Kerberos host — would be uninterpreted by the keystore. Storage and retrieval requests would be authenticated by Kerberos tickets, of course. Only encrypted transfer (KRB_PRIV) should be employed, as insurance against disclosure of such sensitive material.

As noted, the same keystore protocol could be used to supply additional keys for new instances of the same client. For example, a user *pat* could have a separate instance *pat.email*, for receiving encrypted electronic mail. The key for that instance would be restricted to that user, of course.

Generally, transactions with the keystore are initiated by the client. However, there is some question about how to create the additional user keys, as user workstations are not particularly good sources of random keys. The best alternative is to provide a (secure) random number service on the network. When a new client instance is added, this service would be consulted to generate the key; both Kerberos and the keystore would be told about the key.

## SECURITY VALIDATION

Is Kerberos correct? By that we are asking if there are bugs (or trapdoors!) in the design or implementation of Kerberos, bugs that could be used to penetrate a system that relies on Kerberos. Some would say that by making the code widely available, the implementors have enabled would-be penetrators to gain a detailed knowledge of the system, thereby simplifying their task considerably. We reject that notion.

In the late nineteenth century, Kerckhoffs formulated the basic principal under which the security of cryptographic systems should be evaluated: all details of the system design should be assumed to be known by the adversary. Only cryptographic keys specifically assumed to be secret should be unavailable to an attacker.[Kahn67, Kerc83] Given this basic premise, the security of a cryptographic system is evaluated based on concerted efforts at cryptanalysis.

Kerberos is designed primarily as an authentication system incorporating a traditional cryptosystem (the Data Encryption Standard) as a component. Never the less, the philosophy guiding Kerckhoffs' evaluation criterion applies to the evaluation of the security of Kerberos. The details of Kerberos's design and implementation must be assumed known to a prospective attacker, who may also be in league with some subset of servers, clients, and (in the case of hierarchically-configured realms) some authentication servers. Kerberos is secure if and only if it can protect other clients and servers, beginning only with the premise that these client and server keys are secret, and that the encryption system is secure. Moreover, in the absence of a central, trusted ''validation authority'', each prospective user of Kerberos is responsible for judging its security. Of course, a public discussion of system security and publication of security evaluations will facilitate such judgements.

By describing the Kerberos design in publications and making the source code publically available, the Kerberos designers and implementors at Project Athena have made a commendable effort to encourage just such a public system validation. Obviously, this document is itself part of that process. However, the system design and its implementation have undergone significant modification, in part as a consequence of this public discussion. We stress that each modification to the design and implementation results in a new system whose security properties must be considered anew. (Examples of such modifications are the incorporation of hierarchically-organized servers and forwardable tickets in Version 5.)

Hence, on-going modification of Kerberos makes it a moving target for security validation attempts. A detailed security analysis would thus be premature. However, the proposed changes to Kerberos in the next few section are intended, not so much to defeat specific attacks, as to facilitate the validation process. In particular, these suggestions are intended to make Kerberos more modular, in design and implementation. Doing so should make the security consequences of modifications more apparant, and facilitate an incremental approach to Kerberos security validation.

### Message Encoding and Cut-and-Paste Attacks

The most simple analysis of the security of the Kerberos protocols should check that there is no possibility of ambiguity between messages sent in different contexts. That is, a ticket should never be

interpretable as an authenticator, or vice versa. Such an analysis depends on redundancy in the pre-encryption binary encodings of each of the ticket and authenticator information. Currently, that analysis must be repeated with every modification to the protocol. This repetitive and often intricate analysis would be unnecessary if standard encodings (such as ASN.1)[ASN1, BER] were used. These encodings should include the overall message type (such as KRB_TGS_REP or KRB_PRIV). Together with reasonable assumptions about the encryption layer (see the next section), such an encoding scheme would greatly simplify the protocol validation process, particularly as the protocol is modified or extended.

Some use of ASN.1 encodings has been adopted for other reasons in Version 5. We reinforce here that there are design principles other than standards compatibility that motivate such a change.

### The Encryption Layer

Version 4 of Kerberos uses the nonstandard PCBC mode of encryption, *propagating cipher block chaining*, in which plaintext block $i+1$ is exclusive-or'ed with both the plaintext and ciphertext of block $i$ before encryption. This mode was observed to have poor propagation properties that permit message-stream modification: specifically, if two blocks of ciphertext are interchanged, only the corresponding blocks are garbled on decryption. Version 5 replaces PCBC mode with the standard CBC mode, *cipher block chaining*, which exclusive-or's just the ciphertext of block $i$ with the plaintext of block $i+1$ before encryption. A checksum — as of Draft 2, the exact form had not been determined — is used to detect message modification. In order to ensure that duplicate messages have different encryptions, random initial ''confounders'' are added to some message formats. In addition, Version 5 supports alternative encryption algorithms as options.

Both the confounder and checksum mechanisms are meant to augment the security of CBC encryption. They belong in a separate encryption layer, not at the level of the Kerberos protocols themselves. Further, the confounder mechanism should be replaced by using the standard initial vector mechanism of cipher-block chaining.[FIPS81, Davi89]

To prevent message-stream modification during authenticated or private sessions, Version 5 uses a timestamp field to prevent entire encrypted messages from being replayed. This is another concern more properly delegated to the encryption layer, where chaining across the packets of the entire session is the more standard mechanism. (Such chaining avoids both the dependence on a clock and the need to cache recent timestamps.)

Separating the Kerberos protocols from the details of encryption would facilitate both validation of the security of the Kerberos protocols, and implementations and validations involving alternative cryptosystems. Too much focus on mechanism, while endemic to cryptographic protocol design, leads away from the need to state the basic properties required of the encryption layer. We would suggest the following adversarial analysis as the starting point for such a specification: allow an adversary to submit, one after the other, any number of messages for encryption under an unknown key $K$. The adversary also has the ability to take prefixes and suffixes of known messages, exclusive-or known messages, and encrypt or decrypt with known keys. At the end of this process, the adversary should not be able to produce any encrypted messages other than those specifically submitted for encryption. Such an analysis would preclude encryption schemes susceptible to simple chosen-plaintext attacks, as described in a previous section.

Given the intractability of reasoning about DES, or of proving complexity properties of any cryptosystem with bounded key size, such analyses will be no guarantee of overall security. But they can be used to preclude the existence of trivial cut-and-paste attacks.[DeMi83, Moor88]

## RECOMMENDED CHANGES TO THE KERBEROS PROTOCOL

Below, we list our recommended changes to the Kerberos protocol. Our ranking is governed by our estimate of the likelihood and consequences of the attack, balanced against the difficulty of implementing the modification.

a. A challenge/response protocol should be offered as an optional alternative to time-based authentication.
b. Use a standard message encoding, such as ASN.1, which includes identification of the message type within the encrypted data.
c. Alter the basic login protocol to allow for handheld authenticators, in which $\{R\}K_C$, for a random $R$, is used to encrypt the server's reply to the user, in place of the key $K_C$ obtained from the user password. This allows the login procedure to prompt the user with $R$, who obtains $\{R\}K_C$ from the handheld device and returns that value instead of the password itself.
d. Mechanisms such as random initial vectors (in place of confounders), block chaining and message authentication codes should be left to a separate encryption layer, whose information-hiding requirements are clearly explicated. Specific mechanisms based on DES should be validated and implemented.
e. The client/server protocol should be modified so that the multi-session key is used to negotiate a true session key, which is then used to protect the remainder of the session.
f. Support for special-purpose hardware should

be added, such as the keystore. More importantly, future enhancements to the Kerberos protocol should be designed under the assumption that a host, particularly a multi-user host, may be using encryption and key-storage hardware.

g. To protect against trivial password-guessing attacks, the protocol should not distribute tickets for users (encrypted with the password-based key), and the initial exchange should authenticate the user to the Kerberos server.
h. Support for optional extensions should be included. In particular, an option to protect against password-guessing attacks via eavesdropping may be a desirable feature.

**ACKNOWLEDGEMENTS**

We would like to thank D. Davis and T.H. Foregger for their comments on an early draft. We'd especially like to thank C. Neuman for his detailed reviews of many versions of the paper, and his willingness to discuss the issues with us. W. Griffeth helped us with preparation of the appendix on Draft 3. Finally, we'd like to thank the Project Athena and Kerberos development staff for their initial design and implementation of Kerberos, their solicitation of comments, and their responsiveness to our criticisms.

## APPENDIX: VERSION 5 DRAFT 3

Draft 3 has gone a long way towards alleviating our concerns. Many problems have been fixed, and provisions have been made for compatible enhancements to resolve other outstanding issues. These are being refined in ongoing discussion. Still, some issues remain unresolved or unaddressed. In addition, we raise new issues related to older areas of the specification.

In a few places, we mention changes that may be made in future revisions of the specification; the reader is cautioned that these represent our understanding, and only our understanding, of a continuing process.

With one exception, this summary omits areas where the authors' intent was clear or was clarified in private communications. That exception — a way to misuse weak checksums to subvert bidirectional authentication — we include to demonstrate the delicacy inherent in the design and specification of authentication protocols.

### Draft 3 and Our Recommended Changes

We begin by reviewing our recommended changes in light of Draft 3 and subsequent discussions with its authors.

a. The KRB_AS_REQ/KRB_AS_REP and KRB_TGS_REQ/KRB_TGS_REP exchanges now provide challenge/response authentication of the server to the client via a *nonce* field, instead of depending on the workstation time. For application servers, the *e−data* field in the KRB_AP_ERR_METHOD error message can be used by the server to signal the client to use a challenge/response alternative to the time-based kerberos authentication.
b. All encrypted data is labeled with the message type prior to encryption, via full integration of the ASN.1 standard. Although there were many reasons for this decision, we applaud its beneficial impact on security.
c. An optional *padata* field will probably be added to the KRB_AS_REP to allow for handheld authenticator protocol extensions.
d. As discussed, mechanisms such as random initial vectors (in place of confounders), block chaining and message authentication codes are now left to a separate encryption layer, with a much clearer discussion of requirements and of specific mechanisms based on DES.
e. Optional fields will probably be added to the AP_REQ and AP_REP messages to support the negotiation of true session keys.
f. Addition of optional fields (such as *padata*) should facilitate extensions that exploit special-purpose hardware.
g. The initial exchange still does not authenticate the user to the Kerberos server. Thus, the Kerberos equivalent of `/etc/passwd` must be treated as public, and passwords must be chosen and administered with password-guessing attacks in mind. However, the *padata* field facilitates optional implementation of such preauthentication mechanisms.
h. As above, several optional fields facilitate extensions such as exponential-key exchange to protect against password-guessing via eavesdropping.

The following sections discuss some of the revisions in Draft 3 in more detail, and raise some new issues.

### Login Dialog

The login dialog has been enhanced to include an additional authentication data field. This can be used to support hand-held authenticators, pre-encryption of the original request, and future extensions. This is a significant enhancement, but we regret that support for hand-held authenticators and pre-encryption is not yet a part of the standard.

In particular, the optional field in the request message can support some sort of pre-encryption. For example, the nonce field can be sent both in the clear and encrypted in the user's login key, thereby demonstrating that the client is legitimate, and precluding remote collection of tickets encrypted with the user's key. As discussed in the main body of this paper, we feel such a mechanism should be mandatory, not optional. Password-cracking

programs require just this sort of data; there is no need to provide grist for their mill.

As currently released, a challenge-response dialog cannot be implemented by the Draft 3 reply format. While the request message possesses the optional extra field, the reply does not, and hence cannot carry the encrypted key. Adding this field would also permit compatible support of exponential key exchange, wherein each party must send a random exponential. We understand that the optional field will probably be added to the reply.

**The Encryption and Checksum Layers**

There is now a separate, well-defined encryption layer, with specified properties. Among these are that the encryption module be capable of detecting any tampering with the message. The only supported method, in this version, is a CRC-32 checksum sealed within the encrypted portion of the message.

The encryption layer also reaps the benefit of the ASN.1 encoding. Since the encoding includes a length field, it is no longer possible for an attacker to truncate a message, and present the shortened form as a valid encrypted message. If a decision were ever made to replace ASN.1 (say, with something more efficient), this property would need to be preserved.

The confounder has now been moved to the encryption layer, but there is still some confusion of function with the IV used by CBC-mode encryption. As commonly used, an IV *is* a confounder (see, for example, [Voyd83]); to hold it constant during a session negates its purpose and thus requires the additional confounder. We suggest that the IV be used as intended, and be incremented or otherwise altered after each message. Initial values for it should be exchanged during (or derived from) the authentication handshake. Apart from simplifying the definition of the encryption function, this scheme would also allow detection of message deletions by interested applications.

It could be argued that requiring the IV to be handled at a higher layer violates the layering we have espoused. However, an IV is as much an attribute of a cryptosystem as is a key. It would be reasonable to encapsulate the definition of the IV into the definition of the key object passed down to the encryption layer.

The properties required of checksums are not as well-defined. Three types are specified: CRC-32, MD4 and MD4 encrypted with DES.[Rive90] However, no mention is made of their attributes, save that some are labeled ‘‘cryptographic’’. This is a crucial omission, as discussed below. A better classification is whether or not a checksum is ‘‘collision-proof’’, that is, whether or not an attacker can construct a new message with the same checksum. The CRC-32 checksum is not collision-proof, while MD4 is believed to be. Note that encrypting a checksum provides very little protection; if the checksum is not collision-proof and the data is public, an adversary can compute the value and replace the data with another message with the same checksum value. (Several such attacks are indicated below.)

**Weak Checksums and Cut-and-Paste Attacks**

One of the major changes in Draft 3 was the removal of encryption protection from the additional tickets and authorization data that may be enclosed with certain requests. These fields are protected by a checksum sealed in the encrypted authenticator sent with the request. Assume that the checksum algorithm used is CRC-32. (This is permitted by a literal reading of Draft 3, though we have learned that this was not the intent of the authors.) With this assumption, the existence of the ENC-TKT-IN-SKEY option leads to a major security breach, and in particular to the complete negation of bidirectional authentication.

As usual, the client, possessing a valid ticket-granting ticket, sends off a request for a new ticket for some service *S*. The enemy intercepts this request and modifies it. First, the ENC-TKT-IN-SKEY bit is set. This specifies that the ticket, normally encrypted in *S*’s key, should be encrypted in the session key of the enclosed ticket-granting ticket. Second, the attacker’s own ticket-granting ticket is enclosed. Obviously, the attacker knows its session key. Finally, the additional authorization data field is filled in with whatever information is needed to make the CRC match the original version.

Consider what happens. The ticket-granting service, seeing a valid request, sends back a ticket. This ticket, encrypted in the enemy’s key, will not be intelligible to the real service, but of course, it will not get that far. The legitimate client cannot tell that the ticket is misencrypted; tickets are, almost by definition, encrypted in a key known only to the server and Kerberos. When the service is requested, the enemy intercepts the request and unseals the ticket. The client may request bidirectional authentication; however, since the attacker has decrypted the ticket, the session key for that service request is available. Consequently, the bidirectional authentication dialog may be spoofed without trouble.

A number of different factors interacted to make this attack possible. One is obvious: the ticket request was protected by what turned out to be a weak checksum. If a collision-proof checksum were used, the attack would be infeasible; the enemy could not have generated the additional authorization data field necessary to make the new request’s checksum match the original. But there are subtleties here. First, if the additional tickets used

by ENC-TKT-IN-SKEY were encrypted (again), they would have been adequately protected by the very same CRC-32 checksum that was abused in the attack. However, because of the encryption, the enemy would be unable to either discern or match the checksum. In other words, the context is critical; merely refraining from re-encrypting some encrypted data, while using the same checksum to protect it, has led to a security breach. (Note: we have been told that the designers intended to require that the *cname* in the additional ticket match the name of the server for which the new ticket is being requested. This requirement would still permit the intended use of the option, but would foil the attack we describe. Apparently, the requirement was inadvertently omitted from Draft 3.)

A similar attack may be possible using the REUSE-SKEY option. This option was designed for multicast key distribution; with a weak checksum, an attacker can abuse it to generate a service ticket whose key is known. The REUSE-SKEY option also permits a related, albeit less serious, attack. If two tickets, $T1$ and $T2$, share the same key, the attacker can intercept a request for one service, and redirect it to the other. Since the two tickets share the same key, the authenticator will be accepted. Just how damaging this possibility is depends on what sorts of services might want to share the same key. If, say, a file server and a backup server were invoked this way, an attacker might redirect some requests to destroy archival copies of files being edited. A solution to this particular attack is to include either the service name, a collision-proof checksum of the ticket, or both, in the authenticator. To be sure, Draft 3 explicitly warns against using tickets with DUPLICATE-SKEY set for authentication. Servers that obey this restriction are not vulnerable to this attack. Also, we have been told that the REUSE-SKEY option will probably be omitted in future revisions of the protocol.

A last attack of this sort can occur if the attacker substitutes a different ticket for the legitimate one in key distribution replies from Kerberos. The encrypted part of such a message does not contain any checksum to validate that the message was not tampered with in transit. While this appears to be more a denial-of-service attack than a penetration, it would be useful for the client to know this immediately.

Two issues underly this list of potential attacks. As discussed, weak checksums (encrypted but not collision-proof, and over public data) allow an adversary to paste together legitimate-looking messages. Message integrity via strong checksums and/or encryption should be extended to as many protocol messages (and as many fields) as possible.

Second, the REUSE-SKEY and ENC-TKT-IN-SKEY options ‘‘overload’’ the basic protocol, in that tickets may now share session keys or be encrypted in keys other than the service. It is possible that there are other ways an attack could exploit the ensuing ambiguities. These options are intended for very constrained uses, not general authentication; they should not be so intimately integrated into the basic authentication protocol. The same purposes would be served by adding separate message types that cannot be misinterpreted as tickets, and using keys that are derived from but are not identical to those used in the basic protocol.

Even then, an analysis of the final standard is needed, to assure that a minor extension has not negated a security-critical assumption. (E.g., the basic Kerberos protocol assumes that no two tickets share a session key, and that tickets are always encrypted with the server’s key.)

### KRB_SAFE and KRB_PRIV Messages

The KRB_SAFE and KRB_PRIV messages employ the session key distributed with the ticket for integrity-checking and privacy, respectively. Draft 3 dictates that both use time-of-day values to guard against replay, which may be problematic. Currently, the resolution of the timestamp is limited to 1 millisecond, which is far too coarse for many applications. (This and other timestamps in the protocol will probably be changed to microsecond resolution.)

A second problem area is the need for a cache of recently-used timestamps. Obviously, if such messages are used for things like file system requests, the size of the cache could rapidly become unmanageable. Furthermore, if two authenticated or encrypted sessions run concurrently, the cache must be shared between them, or messages from one session can be replayed into the other.

Both problems can be solved if the idea of a timestamp is abandoned in favor of sequence numbers. A random initial sequence number can be transmitted with the authenticator and/or in the KRB_AP_REP message; after each authenticated message is sent, it would, of course, be incremented. The cache is then a simple last-message counter. This mechanism also provides the ability to detect deleted messages, by watching for gaps in sequence number utilization. And, since each session would have its own initial sequence number, it would not be possible for an attacker to perform cross-stream replays, and concurrent access to a common cache is not necessary. (This advantage would be gained even with timestamps if true session keys were used.) It is likely that in a future revision, sequence numbers will be provided as an alternative to the use of timestamps.

### Authenticators

Draft 3 still calls for the use of authenticators to guard against ticket replay. However, there is now a provision for the server to specify that additional authentication is required, and an optional data field for this has been added to the KRB_ERROR reply message. This can be used to implement challenge/response schemes.

The authenticator should have some other fields added to it, some of them optional. As noted earlier, it must contain a collision-proof checksum linking it to the ticket, and an optional initial sequence number. The latter would be used by any applications that might wish to exchange encrypted or authenticated messages.

The authenticator is also the right place to negotiate a true session key. We propose adding a new field for it to both the authenticator and the KRB_AP_REP message. The actual session key could be formed by an exclusive-or of the multisession key associated with the ticket, a randomly-generated field in the authenticator, and a similar field in the reply message. Note that this retains a measure of compatibility with the current scheme: if the two optional fields are not present, the multi-session key will be used as the actual session key.

Negotiation of true session keys, initial sequence numbers, and confounders or IV's could be combined in one standard mechanism, perhaps subsumed as encryption-specific subfields of the session key fields.

### Inter-Realm Authentication

Inter-realm authentication is still problematic. Granted that static configuration files can tell a Kerberos server who its parent is, and even the identities of all of its children, there is still no scalable mechanism to learn of grandchildren or more distant descendants.

To be sure, it is apparently the intention of the authors that the Internet's domain name space be used to denote realms, and — implicitly — the hierarchy of servers. It is far from clear to us that the two hierarchies coincide. Furthermore, such usage is not required. No alternative routing mechanism has been suggested.

Additionally, there are several pieces of the protocol that are unclear or simply do not work with inter-realm tickets. For example, ENC-TKT-IN-SKEY and REUSE-KEY require the ticket-granting server to decrypt a ticket. It cannot do this if the ticket had been issued by another realm. Presumably, of course, the request could be sent to the other realm's ticket-granting server, but it may not possess the necessary key to generate the new ticket.

## NEW RECOMMENDED CHANGES

Below, we include a new list of recommended changes, beyond those we have indicated are likely to be adopted. The first two are repeated from our earlier list, and are now (or will be) implementable as options; we repeat them here to stress our belief that they should be a mandatory part of the protocol.

a. Alter the basic login protocol to allow for challenge/response handheld authenticators.
b. The initial exchange should authenticate the user to the Kerberos server, to complicate password-guessing attacks.
c. Strong checksums, encryption, and additional fields should be used to assure integrity of the basic Kerberos messages. (For example, tickets should be tied more closely to the contexts in which they are used, by including service names in the ticket, and the encrypted part of KRB_AS_REP and KRB_TGS_REP should contain collision-proof checksums of the tickets.)
d. Protocol extensions not related to basic authentication (the ENC-TKT-IN-SKEY and REUSE-SKEY options) should be omitted or use distinct message and ticket formats.

## References

FIPS81. ''DES Modes of Operation,'' Federal Information Processing Standards Publication 81 (December 1980). National Bureau of Standards, U.S. Department of Commerce

ASN1. ''Information Processing Systems — Open Systems Interconnection — Specification of Abstract Syntax Notation One (ASN.1),'' International Standard 8824 (1987). International Organization for Standardization and International Electrotechnical Committee

BER. ''Information Processing Systems — Open Systems Interconnection — Specification of Basic Encoding Rules for Abstract Syntax Notation One (ASN.1),'' International Standard 8825 (1987). International Organization for Standardization and International Electrotechnical Committee

Beke82. H. Beker and F. Piper, *Cipher Systems*, John Wiley & Sons (1982).

Brya88. B. Bryant, *Designing an Authentication System: A Dialogue in Four Scenes*, Draft February 8, 1988.

Davi89. D.W. Davies and W.L. Price, *Security for Computer Networks*, John Wiley & Sons (1989). Second Edition

Davi90. D. Davis and R. Swick, *Workstation Services and Kerberos Authentication at Project Athena*, MIT Laboratory for Computer Science Technical Memorandum 424 (February 1990).

Deav85. C.A. Deavours and L. Kruh, *Machine*

*Cryptography and Modern Cryptanalysis*, Artech House (1985).

DeMi83. R. DeMillo and M. Merritt, ''Protocols for Data Security,'' *Computer* **16**(2) pp. 39-50 (February 1983).

Diff76. W. Diffie and M.E. Hellman, ''New Directions in Cryptography,'' *IEEE Transactions on Information Theory* **6** pp. 644-654 (November, 1976).

Gram84. F.T. Grampp and R.H. Morris, ''Operating System Security,'' *AT&T Bell Laboratories Technical Journal* **63**(8, Part 2) pp. 1649-1672 A&T, (October, 1984).

Kahn67. D. Kahn, *Codebreakers: The Story of Secret Writing*, Macmillan (1967).

Kerc83. A. Kerckhoffs, *La Cryptographie Militaire*, Libraire Militaire de L. Baudoin & Cie., Paris (1883).

Kohl89. J. Kohl, B. Clifford Neuman, and J. Steiner, *The Kerberos Network Authentication Service*, MIT Project Athena (November 6, 1989). Version 5, Draft 2

Kohl90. J. Kohl, B. Clifford Neuman, and J. Steiner, *The Kerberos Network Authentication Service*, MIT Project Athena (October 8, 1990). Version 5, Draft 3

LaMa. B.A. LaMacchia and A.M. Odlyzko, *Computation of Discrete Logarithms in Prime Fields*, (Manuscript in preparation)

Loma89. T.M.A. Lomas, L. Gong, J.H. Saltzer, and R.M. Needham, ''Reducing Risks from Poorly Chosen Keys,'' *Operating Systems Review* **23**(5) pp. 14-18 ACM, (December 1989).

Mill87. S.P. Miller, B.C. Neuman, J.I. Schiller, and J.H. Saltzer, ''Kerberos Authentication and Authorization System,'' in *Project Athena Technical Plan*, (December 1987). Section E.2.1

Mill88. D.L. Mills, ''Network Time Protocol,'' RFC 1059 (July 1988).

Mill89. D.L. Mills, ''Network Time Protocol,'' RFC 1119 (September 1989).

Moor88. J.H. Moore, ''Protocol Failures in Cryptosystems,'' *Proc. IEEE* **76**(5) pp. 594-602 (May 1988).

Morr79. R. Morris and K. Thompson., ''UNIX Password Security,'' *Communications of the ACM* **22**(11) p. 594 (November 1979).

Morr85. R.T. Morris, ''A Weakness in the 4.2BSD TCP/IP Software,'' Computing Science Technical Report No. 117, AT&T Bell Laboratories, Murray Hill, New Jersey (February 1985).

Post80. J.B. Postel, ''User Datagram Protocol.,'' RFC 768 (August 28, 1980).

Post81. J.B. Postel, ''Transmission Control Protocol.,'' RFC 793 (September 1981).

Post83. J.B. Postel and K. Harrenstien, ''Time Protocol.,'' RFC 868 (May 1983).

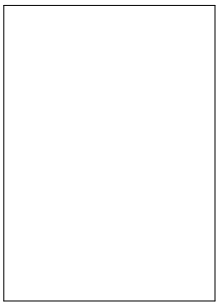
Rive90. R.L. Rivest, ''MD4 message digest algorithm,'' RFC 1186 (October 1990).

Salt90. J.H. Saltzer, private communication June 19, 1990.

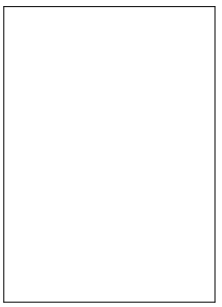
Stei88. J. Steiner, C. Neuman, and J.I. Schiller, ''Kerberos: An Authentication Service for Open Network Systems,'' in *Proc. Winter USENIX Conference*, , Dallas (1988).

Stol88. C. Stoll, ''Stalking the Wiley Hacker,'' *Communications of the ACM* **31**(5) p. 484 (May 1988).

Voyd83. V.L. Voydock and S.T. Kent, ''Security Mechanisms in High-Level Network Protocols,'' *ACM Computer Surveys* **15**(2) pp. 135-171 (June, 1983).



Steven M. Bellovin received a B.A. degree from Columbia University, and an M.S. and Ph.D. in Computer Science from the University of North Carolina at Chapel Hill. While a graduate student, he wrote the original version of *pathalias* and helped create *netnews*. However, the former is not an indictable offense, and the statute of limitations on the latter has expired. Nevertheless, he is still atoning for both actions. He has been at AT&T Bell Laboratories since 1982, where he does research in networks, security, and why the two don't get along. He may be reached electronically as `smb@ulysses.att.com`; those who prefer to murder trees may send scraps of paper to Room 3C-536B, AT&T Bell Laboratories, 600 Mountain Avenue, Murray Hill, NJ 07974, U.S.A.



Michael Merritt received a B.S. from Yale University, and an M.S. and Ph.D. in Information and Computer Science from the Georgia Institute of Technology. His dissertation, "Cryptographic Protocols", developed techniques for exploring security properties of distributed algorithms. He has been at AT&T Bell Laboratories since 1983, where he does research in distributed systems and security. His email address is `mischu@research.att.com`; paper to Room 3D-458, AT&T Bell Laboratories, 600 Mountain Avenue, Murray Hill, NJ 07974, U.S.A.