The Evolution of the *Kerberos* Authentication Service

John T. Kohl

Digital Equipment Corporation

B. Clifford Neuman

Information Sciences Institute
University of Southern California

Theodore Y. Ts'o

Massachusetts Institute of Technology

ABSTRACT

The Kerberos Authentication Service, developed at MIT, has been widely adopted by other organizations to identify clients of network services across an insecure network and to protect the privacy and integrity of communication with those services. While Version 4 was a step up from traditional security in networked systems, extensions were needed to allow its wider application in environments with different characteristics than that at MIT. This paper discusses some of the limitations of Version 4 of Kerberos and presents the solutions provided by Version 5.

1. Introduction

The Kerberos Authentication Service was developed by the Massachusetts Institute of Technology (MIT) to protect the emerging network services provided by Project Athena. Versions 1 through 3 were used internally. Although designed primarily for use by Project Athena, Version 4 of the protocol has achieved widespread use beyond MIT. Models for administration and use of computer services differ from site to site and some environments require support that isn't present in Version 4. Version 5 of the Kerberos protocol incorporates new features suggested by experience with Version 4, making it useful in more situations. Version 5 was based in part upon input from many contributors familiar with Version 4.

This paper begins by describing the Kerberos model and basic protocol exchanges. Section 3 discusses the limitations of Version 4 of Kerberos. The fourth section reviews new features found in Version 5. Section 5 describes the implementation of Version 5 and support for converting existing applications from Version 4. The paper concludes with status and plans for future work.

Terminology and conventions

A *principal* is the basic entity that participates in authentication. In most cases a principal represents a user or an instantiation of a network service on a particular host. Each principal is uniquely named by its *principal identifier*.

Encryption is the process of transforming data into a form that cannot be understood without applying a second transformation. The transformation is affected by an *encryption key* in such a manner that the

This paper is a revision of a paper presented at the Spring 1991 EurOpen Conference, in Tromsø, Norway, and will appear in an upcoming IEEE Computer Society Press book edited by Frances Brazier and Dag Johansen.

The work described here was done while Kohl was at MIT, and in part while Neuman was at the University of Washington.

second transformation can only be applied by someone in possession of the corresponding *decryption key.*

A *secret-key cryptosystem* such as that defined by the Data Encryption Standard (DES) [FIPS46] uses a single key for both encryption and decryption. Such an encryption key is called a *secret* key.

A *public-key cryptosystem* such as RSA [Riv78] uses different keys for encryption and decryption. One of the keys in the pair can be publicly known while the other must be kept private. These keys are referred to as *public* and *private* keys respectively.

Plaintext is a message in its unencrypted form, either before the encryption transformation has been applied, or after the corresponding decryption transformation is complete. *Ciphertext* is the encrypted form of a message, the output of the encryption transformation.

In figures, encryption is denoted by showing the plaintext surrounded by curly braces ({}) followed by a key (K) whose subscript denotes the principals who possess or have access to that key. Thus, "abc" encrypted under c's key is represented as $\{abc\}K_c$.

2. The *Kerberos* Model

Kerberos was developed to enable network applications to securely identify their peers. To achieve this, the client (initiating party) conducts a three-party message exchange to prove its identity to the server (the contacted party). The client proves its identity by presenting to the server a *ticket* (shown in figures as $T_{c,s}$) which identifies a principal and establishes a temporary encryption key that may be used to communicate with that principal, and an *authenticator* (shown in figures as $A_{c,s}$) which proves that the client is in possession of the temporary encryption key that was assigned to the principal identified by the ticket. The authenticator prevents an intruder from replaying the same ticket to the server in a future session.

Tickets are issued by a trusted third party *Key Distribution Center* (KDC). The KDC, proposed by Needham and Schroeder [Nee78], is trusted to hold in confidence secret keys known by each client and server on the network (the secret keys are established out-of-band or through an encrypted channel). The key shared with the KDC forms the basis upon which a client or server believes the authenticity of the tickets it receives. A Kerberos ticket is valid for a finite interval called its *lifetime.* When the interval ends, the ticket expires; any later authentication exchanges require a new ticket from the KDC.

Each installation comprises an autonomously administered *realm* and establishes its own KDC. Most currently-operating sites have chosen realm names that parallel their names under the Internet domain name system (e.g. Project Athena's realm is `ATHENA.MIT.EDU`). Clients in separate realms can authenticate to each other if the administrators of those realms have previously arranged a shared secret.

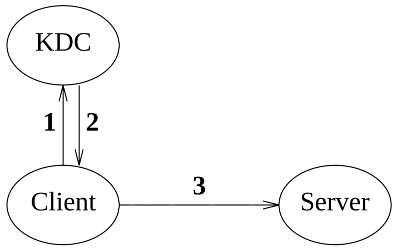
2.1. The initial ticket exchange

Figure 1 shows the messages† required for a client to prove its identity to a server. The basic messages are the same for Versions 4 and 5 of Kerberos though the details of the encoding differ. A typical application uses this exchange when it first establishes a connection to a server. Subsequent connections to the same server require only the final message in the exchange (client caching eliminates the need for the first two messages until the ticket expires).

In the first message the client contacts the KDC, identifies itself, presents a nonce (a timestamp or other non-repeating identifier for the request), and requests credentials for use with a particular server.

Upon receipt of the message the KDC selects a random encryption key $K_{c,s}$, called the *session key,* and generates the requested ticket. The ticket identifies the client, specifies the session key $K_{c,s}$, lists the start and expiration times, and is encrypted in the key K_s shared by the KDC and the server. Because the ticket is encrypted in a key known only by the KDC and the server, nobody else can read it or change the identity of the client specified within it. The KDC next assembles a response, the second message, which it sends to the client. The response includes the session key, the nonce, and the ticket. The session key and nonce are encrypted with the client's secret key K_c (in Version 4 all fields are encrypted in K_c).

† For clarity, the figures show a simplified version of the messages. Other message fields present in the actual messages are less relevant to the present discussion.



1. Client $\rightarrow$ KDC: c, s, n
2. KDC $\rightarrow$ Client: $\{K_{c,s}, n\}K_c, \{T_{c,s}\}K_s$
3. Client $\rightarrow$ Server: $\{A_c\}K_{c,s}, \{T_{c,s}\}K_s$

(In version 4, message 2 is $\{K_{c,s}, n, \{T_{c,s}\} K_s\}K_c$)

Figure 1: *Getting and using an Initial Ticket*

Upon receiving the response the client decrypts it using its secret key (usually derived from a password). After checking the nonce, the client caches the ticket and associated session key for future use.

In the third message the client presents the ticket and a freshly-generated authenticator to the server. The authenticator contains a timestamp and is encrypted in the session key $K_{c,s}$. Upon receipt the server decrypts the ticket using the key it shares with the KDC (this key is kept in secure storage on the server's host) and extracts the identity of the client and the session key $K_{c,s}$. To verify the identity of the client, the sever decrypts the authenticator (using the session key $K_{c,s}$ from the ticket) and verifies that the timestamp is current.

Successful verification of the authenticator proves that the client possesses the session key $K_{c,s}$, which it only could have obtained if it were able to decrypt the response from the KDC. Since the response from the KDC was encrypted in K_c, the key of the user named in the ticket, the server may reasonably be assured that identity of the client is in fact the principal named in the ticket.

If the client requests mutual authentication from the server, the server responds with a fresh message encrypted using the session key. This proves to the client that the server possesses the session key, which it could only have obtained if it was able to decrypt the ticket. Since the ticket is encrypted in a key known only by the KDC and the server, the response proves the identity of the server.

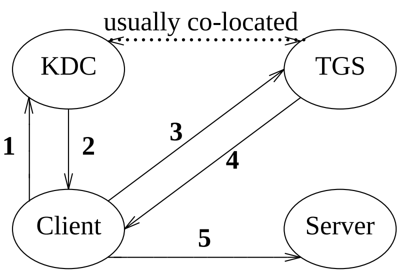
For greater detail on the messages in Version 4 of Kerberos the reader is referred to [Ste88] and [Mil87]. Details about Version 5 can be found in [Koh92].

2.2. The additional ticket exchange

To reduce the risk of exposure of the client's secret key K_c and to make the use of Kerberos more transparent to the user, the exchange above is used primarily to obtain a ticket for a special *ticket-granting server* (TGS). The client erases its copy of the client's secret key once this ticket-granting ticket (TGT) has been obtained,

The TGS is logically distinct from the KDC which provides the initial ticket service, but the TGS runs on the same host and has access to the same database of clients and keys used by the KDC (see Figure 2). A client presents its TGT (along with other request data) to the TGS as it would present it to any other server (in an application request); the TGS verifies the ticket, authenticator, and accompanying request, and replies with a ticket for a new server. The protected part of the reply is encrypted with the session key from the TGT, so the client need not retain the original secret key K_c to decrypt and use this reply. The client then uses these new credentials as before to authenticate itself to the server, and perhaps to verify the identity of the server.

Once the authentication is established, the client and server share a common session key $K_{c,s}$, which has never been transmitted over the network without being encrypted. They may use this key to protect subsequent messages from disclosure or modification. Kerberos provides message formats which an application may generate as needed to assure the integrity or both the integrity and privacy of a message.



1. Client → KDC: c, tgs, n
2. KDC → Client: $\{K_{c,tgs}, n\}K_c, \{T_{c,tgs}\}K_{tgs}$
3. Client → TGS: $\{A_c\}K_{c,tgs}, \{T_{c,tgs}\}K_{tgs}$, s, n
4. TGS → Client: $\{K_{c,s}, n\}K_{c,tgs}, \{T_{c,s}\}K_s$
5. Client → Server: $\{A_c\}K_{c,s}, \{T_{c,s}\}K_s$

(In version 4, message **2** is $\{K_{c,tgs}, n, \{T_{c,tgs}\}K_{tgs}\}K_c$,
and message **4** is $\{K_{c,s}, n, \{T_{c,s}\}K_s\}K_{c,tgs}$)

Figure 2: *Getting a service ticket*

3. Limitations of Version 4

Version 4 of Kerberos is in widespread use, but some sites require functionality that it doesn't provide, while others have a computing environment or administrative procedures that differ from that at MIT. As a result, work on Kerberos Version 5 commenced in 1989, fueled by discussions with Version 4 users and administrators about their experiences with the protocol and MIT's implementation.

3.1. Environmental shortcomings

Kerberos Version 4 was targeted primarily for Project Athena [Cha90], and as such in some areas it makes assumptions and takes approaches that are not appropriate universally:

Encryption system dependence: The Version 4 protocol uses only the Data Encryption Standard (DES) to encrypt messages. The export of DES from the USA is restricted by the U.S. Government, making truly widespread use of Version 4 difficult.

Internet protocol dependence: Version 4 requires the use of Internet Protocol (IP) addresses, which makes it unsuitable for some environments.

Message byte ordering: Version 4 uses a "receiver makes right" philosophy for encoding multi-byte values in network messages, where the sending host encodes the value in its own natural byte order and the receiver must convert this byte order to its own native order. While this makes communication between two hosts with the same byte order simple, it does not follow established conventions and will preclude interoperability of a machine with an unusual byte order not understood by the receiver.

Ticket lifetimes: The valid life of a ticket in Version 4 is encoded by a UNIX timestamp issue date and an 8-bit lifetime quantity in units of five minutes, resulting in a maximum lifetime of 21¼ hours. Some environments require longer lifetimes for proper operation (e.g. a long-running simulation which requires valid Kerberos credentials during its entire execution).

Authentication forwarding: Version 4 has no provision for allowing credentials issued to a client on one host to be forwarded to some other host and used by another client. Support for this might be useful if an intermediate server needs to access some resource with the rights of the client (e.g. a print server needs access to the file server to retrieve a client's file for printing), or if a user logs into another host on the network and wishes to pursue activities there with the privileges and authentication available on the originating host.

Principal naming: In Version 4, principals are named with three components: name, instance, and realm, each of which may be up to 39 characters long. These sizes are too short for some applications and installation environments. In addition, due to implementation-imposed conventions the normal character set allowed for the name portion excludes the period (.), which is used in account names on some systems. These same conventions dictate that the account name match the name portion of the principal identifier, which is unacceptable in situations where Kerberos is being installed in an existing network with non-unique account names.

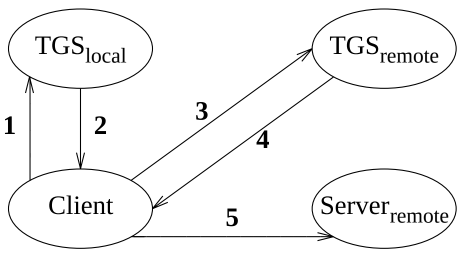
Inter-realm authentication: Version 4 provides cooperation between authentication realms by allowing each pair of cooperating realms to exchange an encryption key to be used as a secondary key for the ticket-granting service. A client can obtain tickets for services from a foreign realm's KDC by first obtaining a ticket-granting ticket for the foreign realm from its local KDC and then using that TGT to obtain tickets for the foreign application server (see Figure 3). This pair-wise key exchange makes inter-realm ticket requests and verification easy to implement, but requires $O(n^2)$ key exchanges to interconnect n realms (see Figure 4). Even with only a few cooperating realms, the assignment and management of the inter-realm keys is an expansive task.

3.2. Technical deficiencies

In addition to the environmental problems, there are some technical deficiencies in Version 4 and its implementation. Bellovin and Merritt [Bel90] provide detailed analyses of some of these issues.

Double Encryption: As shown in Figure 1, the ticket issued by the Kerberos server in Version 4 is encrypted twice when transmitted to the client, and only once when sent to the application server. There is no need to encrypt it in the message from the KDC to the client, and doing so can be wasteful of processing time if encryption is computationally intensive (as will be the case for most software-based encryption implementations; see [Mer90] for discussion of fast software-based encryption methods).

PCBC encryption: Kerberos Version 4 uses a non-standard mode of DES to encrypt its messages. FIPS 81 [FIPS81] describes the normal cipher-block-chaining (CBC) mode of DES. Version 4 uses a modified Version called plain- and cipher-block-chaining mode (PCBC). This mode was an attempt to provide data encryption and integrity protection in one operation. Unfortunately, it allows an intruder to modify a message with a special block-exchange attack which may not be detected by the recipient [Koh89].



1. Client $\rightarrow$ $\mathrm{TGS}_{\mathrm{local}}$: $\{A_c\}K_{c,tgs}$,$\{T_{c,tgs}\}K_{tgs}$, tgs_{rem}
2. $\mathrm{TGS}_{\mathrm{local}}$ $\rightarrow$ Client: $\{K_{c,tgs_{rem}}\}K_{c,tgs}$,$\{T_{c,tgs_{rem}}\}K_{tgs_{rem}}$
3. Client $\rightarrow$ $\mathrm{TGS}_{\mathrm{remote}}$: $\{A_c\}K_{c,tgs_{rem}}$,$\{T_{c,tgs_{rem}}\}K_{tgs_{rem}}$, s_{rem}
4. $\mathrm{TGS}_{\mathrm{remote}}$ $\rightarrow$ Client: $\{K_{c,s_{rem}}\}K_{c,tgs_{rem}}$,$\{T_{c,s_{rem}}\}K_{s_{rem}}$
5. Client $\rightarrow$ $\mathrm{Server}_{\mathrm{remote}}$: $\{A_c\}K_{c,s_{rem}}$,$\{T_{c,s_{rem}}\}K_{s_{rem}}$

(In version 4, message **2** is $\{K_{c,tgs_{rem}}$,$\{T_{c,tgs_{rem}}\}K_{tgs_{rem}}\}K_{c,tgs}$,
and message **4** is $\{K_{c,s_{rem}}$,$\{T_{c,s_{rem}}\}K_{s_{rem}}\}K_{c,tgs_{rem}}$)

Figure 3: *Getting a foreign realm service ticket*

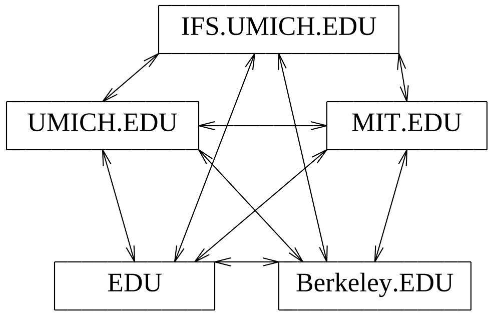


Figure 4: *Version 4 realm interconnections*

Authenticators and replay detection: Kerberos Version 4 uses an encrypted timestamp to verify the freshness of messages and prevent an intruder from staging a successful replay attack. If an authenticator (which contains the timestamp) is out of date or is being replayed, the application server rejects the authentication. However, maintaining a list of unexpired authenticators which have already been presented to a service can be hard to implement properly (and indeed is not implemented in the Version 4 implementation distributed by MIT).

Password attacks: The initial exchange with the Kerberos server encrypts the response with a client's secret key, which in the case of a user is algorithmically derived from a password. An intruder is able to record an exchange of this sort and, without alerting any system administrators, attempt to discover the user's password by decrypting the response with each password guess. Since the response from the Kerberos server includes verifiable plaintext [Lom89], the intruder can try as many passwords as are available and will know when the proper password has been found (the decrypted response will make sense).

Session keys: Each ticket issued by the KDC contains a key specific to that ticket, called a session key, which may be used by the client and server to protect their communications once authentication has been established. However, since many clients use a ticket multiple times during a user's session, it may be possible for an intruder to replay messages from a previous connection to clients or servers which do not properly protect themselves (again, MIT's Version 4 implementation does not fully implement this protection for the KRB_SAFE and KRB_PRIV messages). Additionally, there are situations in which a client wishes to share a session key with several servers. This requires special non-standard application negotiations in Version 4.

Cryptographic checksum: The cryptographic checksum (sometimes called a message authentication code or hash or digest function) used in Version 4 is based on the quadratic algorithm described in [Jue85]. The MIT implementation does not perform this function as described; the suitability of the modified version as a cryptographic checksum function is unknown.

4. Changes for Version 5

Version 5 of the protocol has evolved over the past two years based on implementation experience and discussions within the community of Kerberos users. Its final specification has reached closure, and a description of the protocol is available [Koh92]. Version 5 addresses the concerns described above and provides additional functionality.

4.1. Changes between Versions 4 and 5

Use of encryption

To improve modularity and ease export-regulation considerations for Version 5, the use of encryption has been separated into distinct software modules which can be replaced or removed by the programmer as needed. When encryption is used in a protocol message, the ciphertext is tagged with a type identifier so that the recipient can identify the appropriate decryption algorithm necessary to interpret the message.

Encryption keys are also tagged with a type and length when they appear in messages. Since it is conceivable to use the same key type in multiple encryption systems (e.g. different variations on DES encryption), the key type may not map one-to-one to the encryption type.

Each encryption algorithm is responsible for providing sufficient integrity protection for the plaintext so that the receiver can verify that the ciphertext was not altered in transit. If the algorithm does not have such properties, it can be augmented by including a checksum in the plaintext before encryption. By doing this, we can discard the PCBC DES mode, and use the standard CBC mode with an embedded checksum over the plaintext. It is important to consider the effects of chosen plaintext attacks when analyzing the message integrity properties of candidate encryption algorithms. Some potential weaknesses were found with encryption and checksum methods in initial drafts of the Version 5 protocol [Stu92]. These weaknesses were corrected in subsequent revisions.

Network addresses

When network addresses appear in protocol messages, they are similarly tagged with a type and length field so the recipient can interpret them properly. If a host supports multiple network protocols or has multiple addresses of a single type, all types and all addresses can be provided in a ticket.

Message encoding

Network messages in Version 5 are described using the Abstract Syntax Notation One (ASN.1) syntax [ISO8824] and encoded according to the basic encoding rules [ISO8825]. This avoids the problem of independently specifying the encoding for multi-byte quantities as was done in Version 4. It makes the protocol description look quite different from Version 4, but it is primarily the presentation of the message fields that changes; the essence of the Kerberos Version 4 protocol remains.

Ticket changes

The Kerberos Version 5 ticket has an expanded format to accommodate the required changes from the Version 4 ticket. It is split into two parts, one encrypted and the other plaintext. The server's name in the ticket is plaintext since a server with multiple identities, e.g. an inter-realm TGS, may need the name to select a key with which to decrypt the the remainder of the ticket (the name of the server is bookkeeping information only and its protection is not necessary for secure authentication). Everything else remains encrypted. The ticket lifetime is encoded as a starting time and an expiration time (rather than a specific lifetime field), affording nearly limitless ticket lifetimes. The new ticket also contains a new flags field and other new fields used to enable the new features described later.

Naming principals

Principal identifiers are multi-component names in Kerberos Version 5. The identifier is encoded in two parts, the realm and the remainder of the name. The realm is separate to facilitate easy implementation of realm-traversal routines and realm-sensitive access checks. The remainder of the name is a sequence of however many components are needed to name the principal. The realm and each component of the remainder are encoded as separate ASN.1 `GeneralString`s, so there are few practical restrictions on the characters available for principal names.

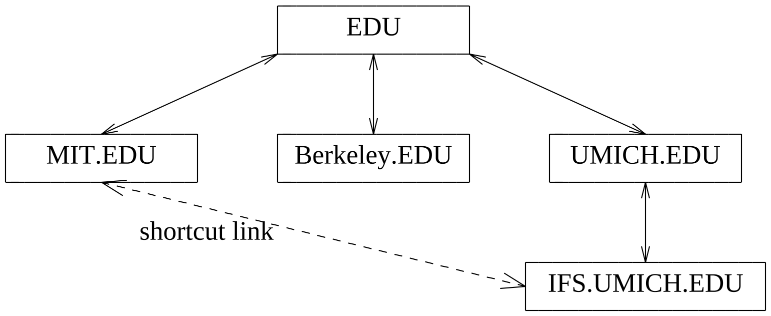


Figure 5: *A Version 5 hierarchy of realms*

Inter-realm support

In Version 5, Kerberos realms cooperate through a hierarchy based on the name of the realm (see Figure 5). A source realm is interoperable with a destination realm if it shares an inter-realm key directly with the destination realm, or if it shares a key with an intermediate realm that is itself interoperable with the destination realm. Each realm exchanges a different pair of inter-realm keys with its parent node and each child. These keys are used in a common encryption system to obtain tickets for each successive realm along the path. This arrangement reduces the number of key exchanges to $O(\log(n))$.

When an application needs to contact a server in a foreign realm, it "walks" up and down the tree toward the destination realm, contacting each realm's KDC in turn, asking for a ticket-granting ticket to the foreign realm. In most cases, the KDC will issue a ticket for the next node in the proper direction on the tree. If a realm has established a "shortcut" spanning link with some realm further in the path, it issues a ticket-granting ticket for that realm instead. This way every realm can interoperate, and heavily-traveled paths can be optimized with a direct link.

When a ticket for the end service is finally issued, it will contain an enumeration of all the realms consulted in the process of requesting the ticket. An application server which applies strict authorization rules is permitted to reject authentication which passes through certain untrusted realms.

4.2. New protocol features in Version 5

In addition to the changes discussed above, several new features are supported in Version 5.

Tickets

Version 5 tickets contain several additional timestamps and a flags field. These changes allow greater flexibility in the use of tickets than was available in Version 4.

Each ticket issued by the KDC using the initial ticket exchange is flagged as such. This allows servers such as a password changing server to require that a client present a ticket obtained by direct use of the client's secret key K_c instead of one obtained using a TGT. Such a requirement prevents an attacker from walking up to an unattended but logged in workstation and changing another user's password.

Tickets may be issued as renewable tickets with two expiration times, one for a time in the near future, and one later. The ticket expires as usual at the earlier time, but if it is presented to the KDC in a renewal request before this earlier expiration time, a replacement ticket is returned which is valid for an additional period of time. The KDC will not renew a ticket beyond the second expiration indicated in the ticket. This mechanism has the advantage that although the credentials can be used for long periods of time, the KDC may refuse to renew tickets which are reported as stolen and thereby thwart their continued use.

A similar mechanism is available to assist authentication during batch processing. A ticket issued as post-dated and invalid will not be valid until its post-dated starting time passes and it is replaced with a validated ticket. The client validates the ticket by presenting it to the KDC as described above for renewable tickets.

Authentication forwarding can be implemented by contacting the KDC with the additional ticket exchange and requesting a ticket valid for a different set of addresses than the TGT used in the request. The KDC will not issue such tickets unless the presented TGT has a flag set indicating that this is a permissible use of the ticket. When the entity on the remote host is granted only limited rights to use the authentication, the forwarded credentials are referred to as a *proxy* (after the proxy used in legal and financial affairs). Proxies are handled similarly to forwarded tickets, except that new proxy tickets will not be issued for a ticket-granting service; they will only be issued for application server tickets.

In certain situations, an application server (such as an X Window System server) will not have reliable, protected access to an encryption key necessary for normal participation as a server in the authentication exchanges. In such cases, if the server has access to a user's ticket-granting ticket and associated session key (which in the case of single-user workstations may well be the case), it can send this ticket-granting ticket to the client, who presents it and the user's own ticket-granting ticket to the KDC. The KDC then issues a ticket encrypted in the session key from the server's ticket-granting ticket; the application server has the proper key to decrypt and process this ticket. The details of such an exchange are presented in [Dav90].

Authorization data

Kerberos is concerned primarily with authentication; it is not directly concerned with the related security functions of authorization and accounting. To support the implementation of these related functions by other services, Version 5 of Kerberos provides a mechanism for the tamper-proof transmission of authorization and accounting information as part of a ticket. This information takes the form of restrictions on the use of a ticket. The encoding of each restriction is not a concern of the Kerberos protocol, but is instead defined by the authorization or accounting mechanism in use. Restrictions are carried in the *authorization data* field of the ticket.

When a ticket is requested, restrictions are sent to the KDC where they are inserted into the ticket, encrypted, and thus protected from tampering. In the protocol's most general form, a client may request that the KDC include or add such data to a new ticket. The KDC does not remove any authorization data from a ticket; the TGS always copies it from the TGT into the new ticket, and then adds any requested additional authorization data. Upon decryption of a ticket, the authorization data is available to the application server. While Kerberos makes no interpretation of the data, the application server is expected to use the authorization data to appropriately restrict the client's access to its resources.

Among other uses, the *authorization data* field can be used in a proxy ticket to create a capability. The client requesting the proxy from the KDC specifies any authorization restrictions in the authorization data, then securely transmits the proxy and session key to another party, which uses the ticket to obtain limited service from an application server. Neuman [Neu91] discusses possible uses of the *authorization data* field in detail.

The Open Software Foundation's Distributed Computing Environment uses the *authorization data* field for the generation of privilege attribute certificates (PACs). Privilege information is maintained by a privilege server. When a PAC is requested by a client the privilege server requests a Kerberos ticket identifying the privilege server itself, but restricting the groups to which the client belongs and specifying a DCE specific user ID. The ticket is then returned to the client which uses it to assert its DCE user ID and prove membership in the listed groups. In essence, the privilege server grants the client a proxy authorizing the client to act as the privilege server to assert the listed DCE user ID and membership in the listed groups. If the ticket did not include restrictions, it would indicate that the client was the privilege server, allowing the client to assert any user ID and membership in any group.

Pre-authentication data

In an effort to complicate the theft of passwords, the Kerberos Version 5 protocol provides fields in the initial- and additional-ticket exchanges to support password alternatives such as hand-held authenticators (devices which have internal circuitry used to generate a continually changing password). In the initial ticket exchange, these fields can be used to alter the key K_c in which the reply is encrypted. This makes a stolen password useless since fresh information from a physical device is needed to decrypt a response. The field can also be used to prove the client's identity to the KDC before any ticket is issued. Doing this makes it a little more difficult for an attacker to obtain a message that can be used to verify password guesses.

This pre-authentication data field is used by the client in the additional ticket exchange to pass the ticket-granting ticket to the KDC; since it is a variable-length array, other values may be sent in the additional-ticket exchange.

Subsession key negotiation

Tickets are cached by clients for later use. To avoid problems caused by the reuse of a ticket's session key across multiple connections, a server and client can cooperate to choose a new *subsession key* which is used to protect a single connection. This subsession key is discarded once the connection is closed.

Negotiation of subsession keys allows an application to protect the privacy of messages broadcast to several recipients. The application can individually negotiate with each recipient to use a common subsession key before beginning the broadcasts.

Sequence numbers

Kerberos provides two message formats for applications to protect their communications. The KRB_SAFE message uses a cryptographic checksum to insure data integrity. The KRB_PRIV message uses encryption to insure integrity and privacy. In Version 4 these messages included as control information a timestamp and the sender's network address. With Version 5, an application may elect to use a timestamp (as before) or a sequence number. If the timestamp is used, the receiver must record the known timestamps to avoid replay attacks; if a sequence number is used the receiver must verify that the messages arrive in the proper order without gaps. There are situations where one choice makes applications simpler (or even possible) to implement; see the discussions in [Koh92].

5. Implementation features

5.1. The base implementation

The MIT implementation of the Version 5 protocols is composed of several run-time libraries with which a program may link. The core library functions will probably be used by all applications; other libraries or subsystems may be replaced or omitted as needed by an application programmer. All code is currently written in "C."

The base functions: The core Kerberos library contains the routines which assemble, disassemble and interpret the network messages. This includes ASN.1 encoding and decoding functions which convert from a machine's native format to the network encoding (currently based on the ISODE package, but another ASN.1 support package may be substituted), routines which verify that requests are answered as expected, and routines to determine which messages are necessary. This core set of routines calls out to the remaining portions of the library as required. A programmer may replace those portions at certain specified interfaces.

Encryption routines: Since multiple encryption types may be in use simultaneously, the core functions call encryption routines through a function table which has entries provided by each encryption system implementation. The core library provides a default cryptosystem table, initialized to list the known encryption types. A programmer may load his own cryptosystem table to replace the default table and avoid linking with the default encryption libraries.

In an attempt to alleviate some possible export restrictions, MIT's implementation distributes its encryption systems separately from the remainder of the system. Only DES is currently available from MIT.

Checksum routines: In a similar fashion to the encryption routines, the core routines call any needed checksum functions through a function table, and compute any necessary sizes based on the information in the table. Certain applications of checksum technology require that the checksum have certain properties. The table entry indicates whether the checksum is keyed (its algorithm is perturbed by an encryption key which cannot be discovered with knowledge only of the algorithm and the checksummed text) and whether the checksum is collision proof (it is computationally infeasible to discover a different checksum text which has the same checksum). The core library provides a replaceable default checksum table.

Four checksums are currently available from MIT: the CRC-32, which is neither keyed nor collision proof (but it is useful for integrity checks within encryption systems); the DES message authentication code (MAC), which is both keyed and collision proof, and MD4 [Riv92a] and MD5 [Riv92b], both of which are collision proof but not keyed.

Credentials cache and key table routines: When clients store tickets and credentials in a cache, the core routines call out through a credentials cache table entry to a separate library module which implements the storing and searching routines for credentials caches. An environment variable can be used to specify the default type and location of a credentials cache, so a user can switch between different types and locations of caches as needed (perhaps to keep the credentials for two roles separate). MIT's implementation provides two credentials cache implementations, one built on C "`stdio`" routines and the other built on UNIX file-descriptor semantics. Other implementations could provide shared-memory or kernel-resident caches.

Servers likewise store their secret keys K_s in key tables accessed by the core routines through a function table. MIT's implementation provides a key table library built on C "`stdio`" routines.

KDC database support: All accesses to the KDC's principal database by the KDC and administrative programs are mediated by a database library which can be replaced if needed. MIT's implementation uses the UNIX *dbm* database system. Since *dbm* does not provide any record or database locking, its use is augmented with separate locking code to mediate between writers and readers. Administrative requests (e.g. adding entries, changing keys or passwords) can be handled on-line.

Operating system support: Although it is targeted for UNIX systems, the MIT implementation is careful to access operating system features only from a few well-contained modules. An operating system support library performs all the accesses required by the rest of the code, such as transmitting and receiving network messages, examining configuration files, checking the system's time-of-day, translating from account names to Kerberos names (and *vice versa*), and performing rudimentary account access checks.

5.2. User interaction

If all parts of Kerberos are working properly, users will not normally be aware that Kerberos authentication is in use by their applications. The normal login process obtains and caches an initial ticket-granting ticket, and applications automatically obtain and cache service tickets as required. Only when authentication fails will users become aware of the underlying use of Kerberos.

If users need to refresh tickets (e.g., if they expire), then they can use the *kinit* program, which will get a new ticket-granting ticket after reading a password from the keyboard. Users examine the cached tickets with *klist* and destroy the cache with *kdestroy*.

When principal names need to be displayed to human users, by convention† they are represented as the sequence of name components separated by slashes (/), followed by an at-sign (@), and the realm name. Thus, a principal with two name components `userX` and `role2` in the realm `ATHENA.MIT.EDU` would be represented as `userX/role2@ATHENA.MIT.EDU`.

Password to key conversion

Since users are not good at remembering binary encryption keys, Kerberos provides routines which convert passwords into keys. The algorithm used to convert a password into an encryption key performs a non-invertible transformation, so that an attacker cannot discover a user's password knowing only K_c. In Version 5, the conversion can be seeded with an additional string (often the realm name) which perturbs the output key, so that a user who is registered in multiple realms and uses the same password in two of those realms will have a different K_c in each realm. Without this perturbation, an attacker discovering the user's key in one realm could impersonate that user in the other realm, without needing to know the user's password. When no additional perturbation string is supplied, the resulting key is the same as the key produced by the Version 4 algorithm.

5.3. Compatibility support for Version 4

There is a small but growing base of Kerberos Version 4 applications, and a number of sites running a Kerberos Version 4 authentication server. MIT's implementation of Version 5 provides several compatibility features which can help sites and programmers convert to Version 5.

Interface compatibility: MIT's implementation of Version 5 includes a "glue library" which presents a Kerberos Version 4 application programming interface (API) but which uses Version 5 protocol messages and routines. This library converts data structures as much as possible between the differing Version 4 and Version 5 data structures. In many cases (especially those that use only a common

† Please note that this is only a *convention,* and other implementations may display the principal names differently.

subset of the Version 4 library functions), an application originally written for Kerberos Version 4 need only be re-linked with this library and the remainder of the Version 5 code to use Version 5 protocols. However, such applications will no longer be compatible with older peer processes, which would still expect the Version 4 messages, and continued maintenance may be made more difficult.

A generic authentication interface: The Generic Security Services API (GSSAPI) [Lin91] is an authentication-system independent programming interface which is currently being developed by the Common Authentication Technology Working Group within the Internet Engineering Task Force. The GSSAPI provides a convenient abstraction boundary for applications writers who wish to take advantage of multiple authentication systems (even ones not yet invented), without needing to be aware of any of the details of those systems. Since the GSSAPI only provides access to those basic authentication services which form a common denominator across different authentication systems, applications which need access to specialized features provided by a particular authentication system will still need to code to that system's native interface. However, the basic functionality to which the GSSAPI provides access should be sufficient for the majority of applications. MIT provides a binding of this interface to the Kerberos Version 5 implementation.

Protocol compatibility: For those sites which wish to convert the Kerberos server to provide the features of Version 5, a compatibility mode may be enabled on the KDC which causes it to accept Version 4 format KDC requests and respond with Version 4 format tickets and messages, as well as accepting Version 5 format requests. This allows an administrator to convert a Version 4 installation to Version 5 slowly, by supporting the old users with the compatibility code. After some grace period, the Version 4 compatibility would be turned off. If a user wishes to use both Version 4 and Version 5 programs simultaneously, the user's key must be encoded using the Version 4 style string-to-key algorithm; the Version 5 response will include information in the pre-authentication data of the ticket response to indicate which string-to-key algorithm should be used by the Version 5 client.

Interface coexistence: The MIT Version 5 libraries were purposely designed to allow an application to simultaneously support both Versions 4 and 5, and this is the suggested compatibility mode. The telnet [Pos83] program distributed with the MIT code can automatically choose an authentication system to use when it connects to a remote system, based on what credentials the user holds and what Versions of authentication the remote telnet server will accept. It implements the current draft specifications of the authentication [Bor92a] and encryption [Bor91] options for both Kerberos Version 5 [Bor92b] and Kerberos Version 4 [Bor92c] authentication systems.

Program compatibility: Another possible compatibility mode can be fabricated by maintaining separate copies of network applications which use Version 4 and Version 5 protocol messages. The user would use a generic name for the application, and the application would try each authentication system in turn, by executing a separate copy of the program for each system (see Figure 6). When authentication is successfully completed, the application would proceed as normal. On both the client and server sides of the application, this approach requires two copies of the same program, each linked with a different authentication system. The different versions of the server would each accept requests at different network ports, and the different clients would only send a request to the server which supports its authentication type.

This approach could be mixed with the glue library and/or single-server approaches, by creating the separate clients using the glue library and/or using a single server program which understands both protocols.

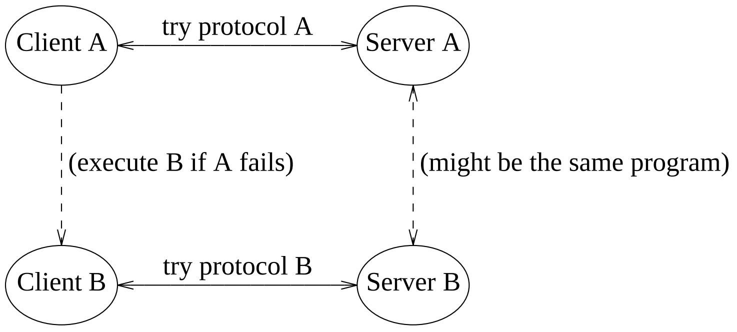


Figure 6: *Implementing protocol compatibility by executing separate programs*

6. Future work

Version 5 of Kerberos is a step toward the design of an authentication system that is widely applicable. We believe the framework is flexible enough to accommodate future requirements. Some items we expect to add to Kerberos in the near future include:

Public-key cryptosystems: The encryption specifications in Kerberos Version 5 are designed primarily for secret-key cryptosystems, but we are considering support for public-key cryptosystems. One advantage of such support will be the ability to interoperate with the evolving certificate infrastructure for Privacy Enhanced Mail. There is also work proceeding on the development of a hybrid Internet Authentication System (IAS) that will provide interoperability between Kerberos and public key based systems such as Digital Equipment Corporation's DASS [Tar91].

"Smartcards": Several companies manufacture hand-held devices which can be used to augment normal password security methods, and there is strong interest within the industry to integrate one or more of these systems with Kerberos. Work is underway to use the pre-authentication data field to pass the additional information needed to use such devices.

In the more distant future it might also be possible to program a smartcard to directly take part in the Kerberos protocol. To do so would require special hardware to support communication between the smartcard and the workstation (so that the smartcard could communicate with the KDC). The advantage of such an approach is that the initial Kerberos exchange could take place without making the user's password available to a potentially untrusted workstation.

Remote administration: The current protocol specifications do not specify an administrative interface to the KDC database. MIT's implementation provides a sample remote administration program which allows administrators to add and modify entries and users to change their keys. We would like to standardize such a protocol. Some features we would like to add include remote extraction of server key tables, password "quality checks," and a provision for servers to change their secret keys automatically every so often.

Database propagation: The current implementation provides reliable KDC service by a periodic bulk-copy of the KDC database to slave KDC machines. It might be more convenient and/or efficient to build the KDC on distributed database technologies. However, to insure that an attacker cannot illegitimately obtain any database entry, the technology must provide for private secure transmission of the database elements to each server,

Validation suites: The current implementation does not include a complete validation suite to verify that the protocol is properly implemented. Such a suite could prevent future security problems in the case of a faulty implementation, and would help facilitate interoperation of diverse implementations.

Applications: There are many network applications that would benefit from the addition of authentication. Among the highly visible examples are electronic mail, popular bulletin-board systems (such as Usenet), and distributed file systems. It is hoped that application designers will consider authentication and related security services when designing their protocols. The generic application programming interface should go a long way toward making it possible to do so.

Acknowledgements

The work described here has been the result of many MIT Project Athena and MIT Network Services staff members' visions, ideas, and hard work.

The authors would especially like to thank Steve Bellovin, Jennifer Steiner, and Ralph Swick for their comments on early drafts of this paper.

REFERENCES

Bel90. S. M. Bellovin and M. Merritt, ''Limitations of the Kerberos Authentication System,'' *Computer Communications Review* **20**(5), pp. 119-132 (October 1990).

Cha90. George A. Champine, Daniel E. Geer, and William N. Ruh, ''Project Athena as a Distributed Computer System,'' *IEEE Computer* **23**(9), pp. 40-50 (September 1990).

Bor91. D. Borman, Editor, ''Telnet Encryption Option,'' Internet-Draft, Internet Engineering Task Force, Telnet Working Group (July 1991).

Bor92a. D. Borman, Editor, ''Telnet Authentication Option,'' Internet-Draft, Internet Engineering Task Force, Telnet Working Group (February 1992).

Bor92b. D. Borman, Editor, ''Telnet Authentication: Kerberos Version 5,'' Internet-Draft, Internet Engineering Task Force, Telnet Working Group (February 1992).

Bor92c. D. Borman, Editor, ''Telnet Authentication: Kerberos Version 4,'' Internet-Draft, Internet Engineering Task Force, Telnet Working Group (February 1992).

Dav90. Don Davis and Ralph Swick, ''Workstation Services and Kerberos Authentication at Project Athena,'' Technical Memorandum TM-424, MIT Laboratory for Computer Science (February 1990).

Jue85. R. R. Jueneman, S. M. Matyas, and C. H. Meyer, ''Message Authentication,'' *IEEE Communications* **23**(9), pp. 29-40 (September 1985).

Koh89. John T. Kohl, ''The Use of Encryption in Kerberos for Network Authentication,'' in *Crypto '89 Conference Proceedings*, International Association for Cryptologic Research, Santa Barbara, CA (August 1989).

Koh92. John T. Kohl and B. Clifford Neuman, ''The Kerberos Network Authentication Service,'' Version 5 Revision 5, Project Athena, Massachusetts Institute of Technology (April 1992).

Lin91. John Linn, ''Generic Security Service Application Program Interface,'' Internet-Draft, Internet Engineering Task Force, Common Authentication Technology Working Group (June 1991).

Lom89. T. Mark A. Lomas, Li Gong, Jerome H. Saltzer, and Roger M. Needham, ''Reducing Risks from Poorly Chosen Keys,'' *Operating Systems Review* **23**(5), pp. 14-18 (December 1989).

Mer90. Ralph C. Merkle, ''Fast Software Encryption Functions,'' in *Crypto '90 Conference Proceedings*, International Association for Cryptologic Research, Santa Barbara, CA (August 1990).

Mil87. S. P. Miller, B. C. Neuman, J. I. Schiller, and J. H. Saltzer, *Section E.2.1: Kerberos Authentication and Authorization System*, M.I.T. Project Athena, Cambridge, Massachusetts (December 21, 1987).

FIPS46. National Bureau of Standards, U.S. Department of Commerce, ''Data Encryption Standard,'' Federal Information Processing Standards Publication 46, Washington, DC (1977).

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

Nee78. Roger M. Needham and Michael D. Schroeder, ''Using Encryption for Authentication in Large Networks of Computers,'' *Communications of the ACM* **21**(12), pp. 993-999 (December, 1978).

Neu91. B. Clifford Neuman, ''Proxy-Based Authorization and Accounting for Distributed Systems,'' Technical Report 91-02-01, Department of Computer Science and Engineering, University of

Washington (March 1991).

Pos83. J. Postel and J. Reynolds, ‘‘TELNET Protocol Specification,’’ RFC 854, University of Southern California, Information Sciences Institute (May 1983).

Riv92a. R. Rivest, ‘‘The MD4 Message Digest Algorithm,’’ RFC 1320, MIT Laboratory for Computer Science (April 1992).

Riv92b. R. Rivest, ‘‘The MD5 Message Digest Algorithm,’’ RFC 1321, MIT Laboratory for Computer Science (April 1992).

Riv78. R. L. Rivest, A. Shamir, and L. Adleman, ‘‘A Method for Obtaining Digital Signatures and Public-Key Cryptosystems,’’ *Communications of the ACM* **21**(2), pp. 120-126 (February 1978). See also U.S. Patent 4,405,829.

ISO8824. International Organization for Standardization, ‘‘Information Processing Systems - Open Systems Interconnection - Specification of Abstract Syntax Notation One (ASN.1),’’ IS 8824 (December 1987). First Edition.

ISO8825. International Organization for Standardization, ‘‘Information Processing Systems - Open Systems Interconnection - Specification of Basic Encoding Rules for Abstract Syntax Notation One (ASN.1),’’ IS 8825 (November 1987). First Edition.

Ste88. J. G. Steiner, B. C. Neuman, and J. I. Schiller, ‘‘Kerberos: An Authentication Service for Open Network Systems,’’ pp. 191-202 in *Usenix Conference Proceedings*, Dallas, Texas (February, 1988).

Stu92. Stuart G. Stubblebine and Virgil D. Gligor, ‘‘On Message Integrity in Cryptographic Protocols,’’ in *Proceedings of the IEEE Symposium on Research in Security and Privacy*, Oakland, California (May 1992).

Tar91. Joseph J. Tardo and Kannan Alagappan, ‘‘SPX: Global Authentication Using Public Key Certificates,’’ in *Proceedings of the IEEE Symposium on Research in Security and Privacy*, Oakland, California (May 1991).

Kohl may be reached at UC Berkeley, Computer Science Division, 571 Evans Hall, Berkeley, CA 94720, USA. Email: jtkohl@cs.berkeley.edu.

Neuman may be reached at USC/ISI, 4676 Admiralty Way, Marina del Rey, CA 90292-6695, USA. Telephone +1 (310) 822-1511. Email: bcn@isi.edu.

Ts’o may be reached at MIT Room E40-342b, 77 Massachusetts Avenue, Cambridge, MA 02139, USA. Email: tytso@mit.edu.