11 KERBEROS V5

This chapter describes Kerberos Version 5, but it assumes you already understand Kerberos V4. So even if you think you're only interested in V5, it's a good idea to read the previous chapter first.

Kerberos V5 represents a major overhaul of Version 4. While the basic philosophy remains the same, there are major changes to the encodings and major extensions to the functionality. The motivation behind the changes in Kerberos V5 is to allow greater flexibility in the environments in which Kerberos can be used.

11.1 ASN.1

ASN.1 [ISO87, PISC93] is a data representation language standardized by ISO. It looks very similar to data structure definitions in programming languages. ASN.1 is popular among spec writers and standards bodies because it gives people a way to precisely define data structures without worrying about different data representations, such as bit and byte order, on different machines.

Where Kerberos V4 messages had a largely fixed layout with variable-length fields marked in an ad hoc way, Kerberos V5 uses ASN.1 syntax with the Basic Encoding Rules (BER), which makes it easy for fields to be optional, of varying length, and tagged with type information to allow future versions to include additional encodings.

This flexibility comes with a price. ASN.1 has a lot of overhead. It adds bytes of overhead in databases and bytes on the wire, and increases the complexity of the code. Often when people define things in ASN.1, they don't realize what the actual format will be. It is possible to use ASN.1 in a way that would create less overhead, but that takes expertise in ASN.1, and there are very few security protocol designers who have any interest in the specifics of ASN.1.

To illustrate the point, in Kerberos V4 an address is four octets. Admittedly this is not sufficiently flexible, since it assumes IP addresses. Had the Kerberos designers custom-designed a more flexible address format, they'd probably have designed something like

- a one-byte address type (defining type codes for IP and perhaps DECnet, CLNP, IPX, and Appletalk)
- a one-byte length, specifying the length in bytes of the address (this is needed because some address types, for instance CLNP, are variable-length)
- and then the address.

With this format it would have taken six bytes to encode an IP address, rather than four in Kerberos V4. However, Kerberos V5 defines an address using the ASN.1 definition:

```
HostAddress ::= SEQUENCE {
            addr-type[0]     INTEGER,
            address[1]       OCTET STRING }
```

What does this mean? It means that an address has two components: **addr-type** and **address**. That sounds reasonable. But what does the Kerberos V5 definition of an address actually expand into? Somehow it adds 11 octets of overhead to each address, meaning that encoding an IP address in V5 requires 15 octets instead of V4's four octets.

Where does this 11 octet overhead come from? The construct **SEQUENCE** requires an octet to specify that it's a sequence, and an octet to specify the length of the entire sequence. The first component, **addr-type[0] INTEGER**, requires four octets of overhead in addition to the single-octet integer that specifies the type of address:

- **addr-type[0]** requires an octet to specify that it's type 0, then an octet to specify the length of what follows.
- **INTEGER** requires a minimum of three octets: an octet to specify that it's an integer (TYPE), an octet to specify how many octets (LENGTH) of integer, and at least one octet giving the value of the integer.

Similarly, the next component, **address[1] OCTET STRING**, requires four octets of overhead in addition to the actual address.

With clever use of ASN.1, it would have been possible to reduce the overhead substantially. For instance, by defining things with **IMPLICIT**, the representations are more compact, because then the TYPE and LENGTH fields do not appear. For example, four bytes of overhead are avoided by defining an address as follows:

```
HostAddress ::= SEQUENCE {
            addr-type[0]     IMPLICIT INTEGER,
            address[1]       IMPLICIT OCTET STRING }
```

The above definition would reduce the overhead from 11 octets to 7 octets. An ASN.1 guru might define an address as follows:

```
HostAddress ::= CHOICE {
    ip_address[0]    IMPLICIT OCTET STRING,
    clnp_address[1]  IMPLICIT OCTET STRING,
    ipx_address[2]   IMPLICIT OCTET STRING,
    ...}
```

That definition produces only two bytes of overhead for an address (provided that the world doesn't invent more than 64 types of address), since it expands into a type (the number in square brackets), followed by a length, followed by the address.

Because ASN.1 is ugly to look at, and since the actual byte-expansion is pretty much irrelevant to any reader of this book, when we describe message formats and data structures in Kerberos V5 we'll just list the contents without being specific about the exact encoding.

In Kerberos V4, the B BIT in all the messages specifies the byte order of multibyte fields. In V5, fields are described using ASN.1 syntax, which defines a canonical format for integers. This is an example where use of ASN.1 hides the complexity of the protocol, since ASN.1 ensures a canonical format.

11.2 NAMES

In Kerberos V4, a principal is named by the three fields NAME, INSTANCE, and REALM, which must each be a null-terminated text string up to 40 characters long (including the null). The size of these fields is too short for some environments, and certain characters (like ".") are illegal inside Kerberos strings but required for account names on some systems. In V5, there are two components: the REALM and the NAME. The NAME component contains a type and a varying number of arbitrary strings, so the purpose served by the V4 INSTANCE field is accomplished by using an extra string in the NAME component. In V4, REALMs are DNS standard names, whereas in V5 they can be DNS standard names or X.500 names, and the syntax allows for other name types as well.

11.3 DELEGATION OF RIGHTS

Delegation of rights is the ability to give someone else access to things you are authorized to access. Usually delegation is limited in time (Alice allows Bob to access her resources only for a specified amount of time), or limited in scope (Alice only allows Bob to access a specified subset

of her resources). Why would Alice want to let Bob use her resources? Alice might want to start up a batch job on some remote node Bob that will run in the middle of the night and need to access many of her files located around the network. Or Alice might be logged into host Bob but then want to log into host Earnest from Bob.

One way delegation could be provided is for Alice to send Bob her master key (in an encrypted message to Bob), allowing him to obtain tickets to whatever resources he might need on Alice's behalf. That is clearly not desirable, since it would allow Bob to forever be able to impersonate Alice. If she knew what resources Bob would need to access on her behalf, she could obtain tickets for those resources in advance and send Bob the tickets and keys (in an encrypted message to Bob). Or if she didn't know all the resources in advance, she could give Bob her TGT and session key, which he could use to obtain specific tickets on Alice's behalf as necessary. Not only are these mechanisms inconvenient and/or insecure and therefore undesirable, but they wouldn't work with Kerberos (either V4 or V5) because the ticket contains a network layer address, and Kerberos insists that a ticket must be used from the specified network layer address. In V4, the network layer address in the ticket is the address from which the ticket was requested.

Kerberos V5 explicitly allows delegation by allowing Alice to ask for a TGT with a network layer address different from hers. As a matter of fact it allows Alice to specify multiple addresses to include (in which case the ticket can be used from any of the specified addresses), or allows Alice to request that no address be included (in which case the ticket can be used from any address). Alice logs in as in Kerberos V4, getting a session key and a TGT with her own network layer address. When she later decides that she needs to allow Bob to act on her behalf, she requests a new TGT from the KDC, but this time specifically says that she'd like the network layer address in the TGT to be Bob's address. The new TGT so obtained is not useable by Alice directly, but can be passed to node Bob (along with the corresponding session key). It is a policy decision by the KDC as to whether to issue tickets with no specified address. It's also a policy decision by services on the network as to whether to accept such tickets.

Kerberos could have provided delegation by removing the network layer address from tickets and TGTs and instead having the network layer address in the authenticator. The Kerberos method has the disadvantage and advantage that Alice has to do an extra interaction with the KDC. It's a disadvantage for performance reasons. But it's an advantage because requiring Alice to do something that lets the KDC know she's delegating to Bob enables the KDC to audit delegation events. In the event of a security compromise, the audit trail will tell which nodes had access to which resources.

Sometimes Alice might know enough and be sufficiently security-conscious to specify the range of rights she wishes to delegate to Bob. Kerberos V5 supports two forms of limited delegation:

- Alice can give Bob tickets to the specific services he will need to access on her behalf (rather than giving him a TGT, which would allow him to request tickets to any services).

- When requesting a ticket or TGT that she intends to give to Bob, Alice can request that a field AUTHORIZATION-DATA be added to the ticket or TGT. The field is not interpreted by Kerberos, but is instead application-specific, which means it is left up to the application to define and use the field. The intention is that the field specifies to the application restrictions on what Bob is allowed to do with the ticket. If the field is in a TGT Alice gives to Bob, the field will be copied by the KDC into any ticket Bob gets using that TGT. OSF/DCE security (see §17.7 *DCE Security*) makes extensive use of this field.

Because there is not universal agreement that allowing delegation is always a good idea, Kerberos V5 makes it optional. A flag inside a TGT indicates whether a request for a TGT or ticket with a different network layer address should be allowed. The Kerberos protocol itself does not specify how the KDC should know how to set the various permission flags when creating an initial TGT. One method is the method the MIT implementation chose, which is to configure instructions for setting the permissions into the user's entry in the KDC database. Another possible way of deciding how to set various flags inside the TGT would be to ask the user Alice when she logs in, but that leaves the potential for a horrible user interface: Name_____ Password_____ Proxiable? (Y/N)__ Allow postdated? (Y/N)__ Forwardable? (Y/N)__ Renewable? (Y/N)__ Aisle/Window__ Smoking/Nonsmoking__ ....

There are two flags in a TGT involving delegation permission. One indicates that the TGT is **forwardable**, which means that it can be exchanged for a TGT with a different network layer address. This gives Alice permission to give Bob a TGT, with which Bob can request tickets to any resources on Alice's behalf. When Alice uses a forwardable TGT to request a TGT to be used from Bob's network layer address, she also specifies how the FORWARDABLE flag should be set in the requested TGT. If she requests that the TGT have the FORWARDABLE flag set, then Bob will be able to use that TGT to obtain a TGT for some other entity Carol, allowing Carol to act on Alice's behalf.

The other flag indicates that the TGT is **proxiable**, meaning that it can be used to request tickets for use with a different network layer address than the one in the TGT. This gives Alice permission to get tickets that she can give to Bob, but not a TGT for use by Bob. The Kerberos documentation refers to tickets Alice gives to Bob for use on her behalf as **proxy tickets**.

TGTs have a FORWARDED flag. Tickets have a FORWARDED flag and a PROXY flag. A TGT given to Bob by Alice is marked forwarded. The FORWARDED flag will also be set in any ticket Bob obtains using a TGT marked forwarded. A ticket given to Bob by Alice is marked proxy. The reason for marking tickets in this way is that some applications may want to refuse to honor delegated tickets, and need to recognize them as such. Note that allowing both the KDC and applications to make decisions on whether delegation is allowed makes for a flexible but confusing access control model.

11.4 TICKET LIFETIMES

In V4, the maximum lifetime of a ticket was about 21 hours, since the time in a ticket was encoded as a four-octet start time and a one-octet lifetime (in units of 5 minutes). This was too short for some applications. In Kerberos V5, tickets can be issued with virtually unlimited lifetimes (the farthest in the future that can be specified with a V5 timestamp is Dec 31, 9999). The timestamp format is an ASN.1-defined quantity that is 17 bytes long. Although it has a virtually unlimited lifetime (unlike the V4 timestamp), it is only in seconds, and Kerberos V5, in some cases, would have preferred time expressed down to microseconds. As a result, much of the time when Kerberos V5 passes around a timestamp it also passes around a microsecond time, which is an ASN.1 integer whose representation requires sufficiently many bytes to express the value (in this case one to three bytes, since 999999 is the biggest value), plus a type and length.

Long-lived tickets pose serious security risks, because once created they cannot be revoked (except perhaps by invalidating the master key of the service to which the ticket was granted). So V5 has a number of mechanisms for implementing revocable long-lived tickets.

These mechanisms involve use of several timestamp fields in tickets (and TGTs). Each timestamp is encoded, using glorious ASN.1 format, in 17 octets of information. First we'll give the names of the timestamps, and then we'll explain how they're used:

- START-TIME—time the ticket becomes valid.
- END-TIME—time the ticket expires.
- AUTHTIME—time at which Alice first logged in, i.e. when she was granted an initial TGT (one based on her password). AUTHTIME is copied from her initial TGT into each ticket she requests based on that TGT.
- RENEW-TILL—latest legal end-time (relevant for renewable tickets, see below).

11.4.1 Renewable Tickets

Rather than creating a ticket valid for say, 100 years, the KDC can give Alice a ticket that will be valid for 100 years, but only if she keeps renewing it, say once a day. Renewing a ticket involves giving the ticket to the KDC and having the KDC reissue it. If there is some reason to want to revoke Alice's privileges, this can be done by telling the KDC not to renew any of Alice's tickets.

The KDC is configured with a maximum validity time for a ticket, say a day. If there is a reason for Alice's ticket to be valid for longer than that time, then when Alice requests the ticket, the KDC sets the RENEWABLE flag inside the ticket. The RENEW-TILL time specifies the time beyond which the ticket cannot be renewed.

In order to keep using the ticket, Alice will have to keep renewing it before it expires. If she is ever late renewing it, the KDC will refuse to renew it. Why did Kerberos choose to do it that way? It seems somewhat inconvenient. Node Alice has to keep a demon running checking for tickets that will expire soon, and renew them. If Kerberos allowed renewal of expired tickets, then Alice could wait until she attempts to use a ticket and gets an error message indicating the ticket expired, and then renew it. This would be more convenient. The reasoning in Kerberos is that if Alice could present a ticket a long time after it expired, then if the KDC has been told to revoke the ticket it would have to remember the revoked ticket until that ticket's RENEW-TILL time. As it is, it just has to remember the revoked ticket for a maximum validity time.

The END-TIME specifies the time at which the ticket will expire (unless renewed). When Alice gives the KDC a renewable ticket and requests that it be renewed, the KDC does this by changing END-TIME to be the maximum ticket lifetime as configured into the user's entry in the KDC database, added to the current time (but not greater than RENEW-TILL).

11.4.2 Postdated Tickets

Postdated tickets are used to run a batch job at some time in the future. Suppose you want to issue a ticket starting a week from now and good for two hours. One possible method is to issue a ticket with an expiration time of one week plus two hours from the present time, but that would mean the ticket would be valid from the time it was issued until it expired. Kerberos instead allows a ticket to become valid at some point in the future. Kerberos does this by using the START-TIME timestamp, indicating when the ticket should first become valid. Such a ticket is known as a **postdated ticket**.

In order to allow revocation of the postdated ticket between the time it was issued and the time it becomes valid, there's an INVALID flag inside the ticket that Kerberos sets in the initially issued postdated ticket. When the time specified in START-TIME occurs, Alice can present the ticket to the KDC and the KDC will clear the INVALID flag. This additional step gives the opportunity to revoke the postdated ticket by warning the KDC. If the KDC is configured to revoke the postdated ticket, the validation request will fail.

There's an additional flag inside the ticket, the POSTDATED flag, which indicates that the ticket was originally issued as a postdated ticket. An application could in theory refuse to accept such a ticket, but none currently do and we can't imagine why any applications would care.

A flag, MAY-POSTDATE, which appears in a TGT, indicates whether the KDC is allowed to issue postdated tickets using this TGT.

11.5 KEY VERSIONS

If Alice holds a ticket to Bob and then Bob changes his key, Kerberos enables Alice's ticket to work until it expires by maintaining multiple versions of Bob's key, and tagging Bob's key with a version number where necessary for the KDC or for Bob to know which key to use.

In the KDC database, each version of Bob's key is stored as a triple: ⟨*key*, *p_kvno*, *k_kvno*⟩. *key* is Bob's key encrypted according to the KDC's key. *p_kvno* is the version number of this key of Bob's (*p_* stands for *principal*). *k_kvno* is the version number of the KDC's key that was used to encrypt *key*, since the KDC might also have changed its key recently (*k_* stands for *KDC*).

If Alice asks for a ticket to Bob, the KDC encrypts the ticket with the key for Bob with the highest *p_kvno*. In V4, the KDC did not keep track of more than one key for Bob. It was up to Bob to keep track of all his keys for a ticket expiration interval, in order for Bob to honor unexpired tickets issued with his old key. So why does the KDC need to keep track of multiple keys for Bob in V5? It is because of renewable tickets and postdated tickets. If Alice has a renewable ticket to Bob, and Bob changed his key since the ticket was originally issued, the KDC needs to be able to decrypt the ticket, so it needs to have stored the key with which that ticket was encrypted. When the KDC renews the ticket, it will issue the renewed ticket with the most recent key for Bob. That way the KDC and Bob can forget old key version numbers after a predictable, reasonably small time (like a day) (except for postdated tickets, which is somewhat of a design flaw in Kerberos).

11.6 MAKING MASTER KEYS IN DIFFERENT REALMS DIFFERENT

Suppose Alice is registered in different realms, and suppose Alice is human. Given that humans have a limited capacity for remembering passwords, Alice might wish to have a single password in all the realms in which she is registered. This means that if an intruder discovers her master key in one realm, he can impersonate her in the other realms as well.

In Kerberos V5, the password-to-key conversion hash function uses the realm name. This means that the function, given the same password, will come up with a different master key if the name of the realm is different. The function is such that it is not possible to derive the master key in realm FOO even if the master key derived from the same password in realm BAR is known.

This does not protect against an intruder who manages to obtain Alice's password. This just helps in the case where Alice has chosen a good password and an intruder manages to steal a KDC database from some realm. Stealing that database will allow the intruder to impersonate Alice in that realm, but not to impersonate Alice in any other realms for which she is using the same password. Note that stealing the database will also allow an intruder to mount an off-line password-

guessing attack, and if the password-guessing attack succeeds, then the intruder *can* impersonate Alice in other realms for which she is using the same password.

11.7 OPTIMIZATIONS

There were certain fields in Kerberos V4 that were not necessary and were taken out in V5. In particular, encryption is expensive (especially when done in software), so it is undesirable to unnecessarily encrypt information. In Kerberos V4, a ticket is included in the CREDENTIALS portion of an AS_REP, and the entire CREDENTIALS field, including the ticket, is encrypted. A ticket is already an encrypted message. There is no reason to encrypt the ticket an additional time. (It had better not be necessary—tickets are later sent across the network unencrypted.)

An example of a field that was removed in Kerberos V5 because it was only slightly useful (if at all) was the name of the ticket target inside a ticket; that is, if Alice gets a ticket to Bob, then Bob's name is in the V4 ticket to Bob, but not in the V5 ticket.

11.8 CRYPTOGRAPHIC ALGORITHMS

Kerberos V4 assumes DES is the encryption algorithm. There are two problems with DES. One is that it is not secure enough for high-security environments. The other is that it is considered by the U.S. government to be too secure to export. Kerberos V5 is designed in a modular way which allows insertion of different encryption algorithms. When encryption is used, there is a type field allowing the receiver to know which decryption algorithm to use.

Since different encryption systems use different-length keys, and since some encryption systems allow variable-length keys, in V5 keys are tagged with a type and length.

DES continues to be used in all actual implementations of Kerberos (to our knowledge). Two cryptographic weaknesses in Kerberos V4 (modified Jueneman checksum, which was used for integrity protection without encryption, and PCBC, which was used for encryption and integrity protection) were repaired.

11.8.1 Integrity-Only Algorithms

The modified Jueneman checksum used in Kerberos V4, while never (publicly) broken, was not considered sufficiently secure (see §10.10 *Encryption for Integrity Only*). So in V5 it was replaced by a choice of algorithms.

Why did V5 not simply choose one known-to-be-secure integrity protection algorithm? No algorithm is ever known to be secure. It's just not known to be broken. So V5 selected a few algorithms, with the intent that if a serious cryptographic flaw was found in one of the algorithms being used, a different one could be substituted without changing the rest of the implementation. Unfortunately, if a recipient does not accept all defined algorithms, there is a possibility of non-interoperability (acceptable algorithms are not negotiated). Another problem with having a choice of algorithms is that Kerberos is really only as secure as the weakest algorithm the recipient will accept rather than the strongest. The reason for this is that if one algorithm is weak, then even if your implementation does not transmit it, a forger could use the weak algorithm to impersonate you to any implementation which accepts it.

If Kerberos V5 were designed today, the algorithms of choice would probably be DES-CBC and one or more of the MD algorithms computed on a concatenation of the secret key with the message. However, the idea of using MD(secret|message) as an integrity check wasn't well known at the time. Kerberos V5 does something probably equivalent in terms of security, but harder to explain. Much harder to explain, as a matter of fact. We agonized as to whether to bother you with the details. The algorithms are baroque and technically uninteresting. There never would be a reason to implement them except to be compatible with a Kerberos V5 implementation. But in the interest of completeness, we'll explain them here.

Kerberos V5 documentation refers to an integrity check as a *checksum*. We prefer the term MIC (message integrity code). The MICs specified in V5 are as follows, using the names in the Kerberos documentation. Three of them are required to be supported by implementations. The other two are optional.

- rsa-md5-des (required)
- des-mac (required)
- des-mac-k (required)
- rsa-md4-des (optional)
- rsa-md4-des-k (optional)

11.8.1.1 rsa-md5-des

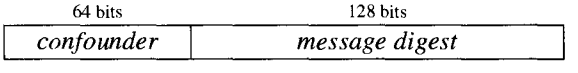
This MIC is one of the required ones. The name is not particularly helpful, except that it's a combination of md5 and des. It has nothing to do with RSA other than that RSADSI (the com-

pany) owns rights to MD5, which is freely distributable provided that RSADSI is credited with every mention of it (or some such legalism).

The way the MIC is calculated is as follows:

1. Choose a 64-bit random number, known as a **confounder**
2. Prepend it to the message:

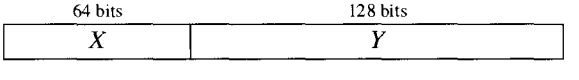
3. Calculate the MD5 message digest of the result, getting a 128-bit quantity.
4. Prepend the confounder chosen in Step 1 to the message digest:



5. Calculate a modified key by taking the KDC-supplied shared secret key and ⊕ing it with $F0F0F0F0F0F0F0F0_{16}$. Call the result K'.
6. Encrypt the result, using DES in CBC mode, using K' and an IV (initialization vector) of 0, resulting in a 192-bit encrypted quantity. That 192-bit quantity is the MIC.

How is this MIC verified? It's actually quite straightforward. You just reverse all the steps.

1. Calculate the modified key, by performing Step 5 above (⊕ing the KDC-supplied shared secret key with $F0F0F0F0F0F0F0F0_{16}$ to get K').
2. Decrypt the MIC, using K' in CBC mode, resulting in a 192-bit quantity. Let's call the first 64 bits of the decrypted quantity X, and the remainder Y:



3. The first 64 bits of the result (X) should be the confounder. To verify that, append X to the message, and calculate the MD5 message digest of the result.

4. If the 128-bit result matches Y, then the MIC is verified as valid.

11.8.1.2 des-mac

This is another of the required MICs. To calculate it do the following:

1. Choose a 64-bit random number, known as a **confounder**.
2. Prepend it to the message:

confounder	*message*

3. Calculate the DES CBC residue of the result (confounder prepended to the message) using the unmodified KDC-supplied shared secret key K, and using an IV of 0. The result is a 64-bit quantity we'll call R, for the *Residue*.
4. Calculate the modified key $K' = K \oplus \text{F0F0F0F0F0F0F0F0}_{16}$.
5. Prepend the 64-bit confounder C to the 64-bit residue R, getting a 128-bit value.
6. Perform DES encryption in CBC mode on the 128-bit $C|R$ from the previous step, using K' as the key, and an IV of 0.
7. The result is the 128-bit MIC.

Verifying this MIC is straightforward (see Homework Problem 9).

11.8.1.3 des-mac-k

This is another of the MICs which are required. The MICs that end with "-k" in their name are the old-style ones, before it occurred to the Kerberos designers that using a modified key would be a good idea. These are no longer recommended, but need to be implemented for backward compatibility.

This MIC is calculated by doing a CBC-residue over the message using the original key K, and using K also as the IV. The MIC is verified the same way.

11.8.1.4 rsa-md4-des

This MIC is the same as rsa-md5-des, except that MD4 is used instead of MD5.

11.8.1.5 rsa-md4-des-k

This MIC is no longer recommended, and is only there for backward compatibility. Again, the "-k" in the name indicates that it was designed before the Kerberos designers realized it would be a good idea to use a modified version of the key for calculating the MIC.

This MIC is calculated as follows. First calculate MD4 of the message, yielding 128 bits (16 octets). Take the result and encrypt it using DES in CBC mode, with the unmodified session key K used as both the encryption key and the IV. The 128-bit result of the encryption is the MIC.

11.8.2 Encryption for Privacy and Integrity

The algorithms in this section provide encryption and integrity protection. The idea is to have an algorithm that not only encrypts the data, but allows Kerberos, when decrypting it, to detect if the message has been altered since being transmitted by the source.

The three algorithms are known in the Kerberos documentation as des-cbc-crc, des-cbc-md4, and des-cbc-md5. The basic idea is that a checksum is combined with the message, and then the message is encrypted with DES in CBC mode. The algorithms use the checksums CRC-32, MD4, and MD5, respectively. All the algorithms do the following:

1. Choose a 64-bit random number known in the Kerberos documentation as a **confounder**.
2. Create the following data structure, where the field CHECKSUM is filled with zeroes and is of the right length for the checksum algorithm of choice (32 bits for des-cbc-crc and 128 bits for the others):

confounder	*checksum*	*message*

3. Calculate the appropriate checksum over the above data structure.
4. Fill in the result in the CHECKSUM field
5. Add enough padding to make the data structure an integral number of 64-bit chunks:

confounder	*checksum*	*message*	*padding*

6. Encrypt the result using DES in CBC mode with an IV of 0.

11.9 HIERARCHY OF REALMS

In Kerberos V4, in order for principals in realm *A* to be authenticated by principals in realm *B* it was necessary for *B*'s KDC to be registered as a principal in *A*'s KDC. For full connectivity, this means that if there are *n* realms, the KDC in each realm has to be registered as a principal in each of the other $n-1$ realms. This is increasingly nightmarish as *n* gets large (see §7.7.4.1 *Multiple KDC Domains*).

In Kerberos V5, it is allowable to go through a series of realms in order to authenticate. For instance, a principal in realm *A* might wish to be authenticated by a principal in realm *C*. However, realm *C* might not be registered in *A*. But perhaps realm *B* is registered in *A*, and realm *C* is regis-

tered in *B*. A principal in *A* can get a ticket for something in *C* by first getting a ticket for *B*, and then asking *B* for a ticket to the KDC in *C*.

By allowing realm *B* to act as intermediary between realm *C* and other realms, we give the KDC at *B* the power to impersonate anyone in the world. Kerberos fixes this vulnerability somewhat by including in tickets a TRANSITED field which lists the names of all the realms that have been transited to obtain the ticket.

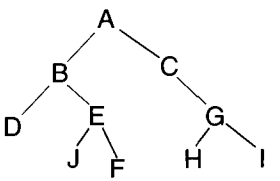
Why is the TRANSITED field useful? Suppose Woodward@Washington-Post.Com is contacted with a ticket that indicates the ticket was issued to the principal named Deep-Throat@WhiteHouse.gov, with the TRANSITED field indicating KGB.Russia. It is possible that Woodward should not assume the party using the ticket is really Mr. or Ms. Throat, since it would be in the interest of and the ability of the owner of the KGB realm's KDC to create a ticket that claims the source is anything. The KGB KDC can give such a ticket, along with the corresponding session key, to a confederate. Or the KDC can use the ticket and session key to impersonate the named source directly.

The only thing the KGB KDC cannot do is avoid being named inside the ticket, since a KDC will reject a ticket if the final entry in the TRANSITED field doesn't match the key with which the ticket is encrypted. If Alice gives Bob a ticket, Bob knows which KDC issued the ticket (it's the one with which he shares the key used to encrypt the ticket). But for all the other information in the ticket (like Alice's name and the other realms mentioned in the TRANSITED field), Bob has to trust the KDC which issued the ticket. And although the KDC which issued the ticket to Bob might be trustworthy, if there's any KDC in the path that isn't, all the earlier realms mentioned in the TRANSITED field and the original principal's name (Alice) are suspect.

The TRANSITED field in the ticket gives enough information for Bob (the service being accessed with the ticket) to know whether there are any realms on the path that Bob considers untrustworthy. A realm might be considered completely untrustworthy as a transit realm, but trustworthy when it claims to be acting on behalf of principals in its own realm. Each principal will have its own policy for which realms to trust.

You could say that by doing this, Kerberos is permitting maximum flexibility in possible policies. Or you could say that Kerberos is abdicating responsibility for this crucial decision by throwing it to the whim of application developers who will almost certainly get it wrong.

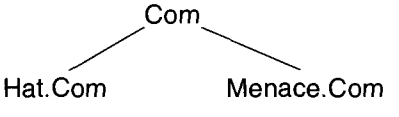
Either way, some sort of policy is necessary. One such policy—and a likely one at that—is to arrange realms into a tree such that each realm shares a key with each of its children and with its parent. The set of realms trusted for any authentication is the shortest path through the tree, i.e., the



path that gets no closer to the root than the common ancestor lowest in the tree.

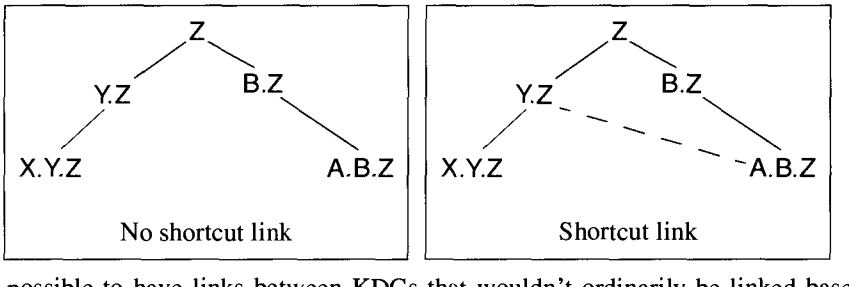
For example, in the above diagram, realm G shares a key with its parent realm (C) and each of its children (H and I). To get from realm I to realm H, you'd go through G. To get from realm F to realm D, you'd go through the lowest common ancestor (B), and to get there you'd have to go through E, so the path would be F–E–B–D.

It's especially convenient if the path of realms can be identified solely on the basis of the syntax of names. If realm names were just unstructured strings, it would be difficult to find a path. Luckily realm names in all current implementations of Kerberos are hierarchical, since they follow either internet or X.500 naming. For instance, assume Cat@Hat.Com wishes to access Dennis@Menace.Com. Cat@Hat.Com resides in realm Hat.Com. Dennis@Menace.Com resides in Menace.Com. The next level of hierarchy is simply called Com. If we create a realm named Com that shares a key with all realms with names of the form *x*.Com, it can then serve as an authentication intermediary. In general, to get from one realm to another, one travels upward to a common ancestor, and then downward to the destination realm.



It is likely that some administrative entity exists which would be a likely CA operator for the Com realm, because some such entity must ensure that there are no name collisions in the .Com space.

Sometimes it might be desirable to shortcut the hierarchy. This might be for efficiency reasons (so authentication between two realms distant in the naming hierarchy does not need to be done via a long sequence of KDCs), or for trust reasons (there might be KDCs along the naming hierarchy path that the two realms would prefer not to have to trust).



It is possible to have links between KDCs that wouldn't ordinarily be linked based on the naming hierarchy. Such links are usually called **cross links**. A safe rule with cross links is that when traversing the naming hierarchy to get to the target, cross links should always be used if they make the path shorter, because it means fewer KDCs need to be trusted.

There are two issues with realm paths. One is how the initiator finds a realm path to the target. As we've shown, if names are hierarchical and the path of realms follows the same hierarchy, with the possible addition of cross links, it is easy to find a path. The other issue is how the target decides whether the realm path used was acceptable. As we said, Kerberos leaves it up to the application.

The TRANSITED field lists the sequence of transited realms, omitting the source and destination realms. Realm names are listed separated by commas. Since the list of realms might get large, Kerberos permits various abbreviations. If the realm list is empty, no realms were transited. But if the realm list consists of a single comma, it means that the hierarchy of realms was transited in the normal way (parent to parent from the source up to the first common ancestor, then child to child down to the destination). Two consecutive commas in a list (or a leading or trailing comma) indicate that the hierarchy was transited in the normal way between the two realms surrounding the comma pair (or between source realm and first-listed realm, or between last-listed realm and destination realm). There are other abbreviation rules as well.

11.10 EVADING PASSWORD-GUESSING ATTACKS

With Kerberos V4, there is no authentication of the request to the KDC for a TGT. Anyone can send a cleartext message to the KDC requesting a TGT for user Pope@Vatican.Com, and the KDC will send back a ticket, encrypted according to Pope's master key. Since the function that maps a password string to a DES key is publicly known, an intruder can use the encrypted credentials for an off-line password-guessing attack to find Pope's password.

To avoid this attack, a mechanism has been added to Kerberos V5 in which information known as PREAUTHENTICATION-DATA can be sent along with the request for a TGT for user Pope which proves that the requester knew user Pope's master key. The preauthentication data consists of a current timestamp encrypted with user Pope's master key.

There's another opportunity for password guessing. Although the preauthentication data forces Alice to prove she knows user Pope's master key before she can obtain a TGT for Pope, she can use her own TGT or master key to ask for a ticket to the principal named Pope. She'll get back a quantity (the ticket to user Pope encrypted according to Pope's master key) which she can use for an off-line password-guessing attack to find Pope's password. Kerberos prevents this attack by marking database entries for human users (such as Pope), with a flag indicating that the KDC should not issue a ticket to this principal. This prevents someone from obtaining a ticket for something whose master key is derived from a password (and therefore vulnerable to password guessing). If, in the future, Kerberos is used for an application where it might make sense to create a ticket to a human user (for instance, electronic mail), then some other mechanism would need to be

devised to prevent Trudy from guessing passwords based on tickets she requests (see Homework Problem 5).

This does not avoid password-guessing attacks completely. Someone can still guess passwords by constructing a request to the KDC for each password guess, and eventually one will be accepted. If passwords are even moderately well chosen, however, this is likely to be a very time-consuming task. Furthermore, a KDC could include code to record the frequency of wrong password guesses and lock the target account and/or alert an administrator should a threshold be exceeded. A more important attack is that an eavesdropper who sees the initial Kerberos login exchange can perform an off-line password guessing attack using either the preauthentication data provided by the user or the TGT sent in response.

11.11 KEY INSIDE AUTHENTICATOR

Suppose Alice wants to have two separate conversations with Bob. If she uses the same key (the Alice-Bob session key chosen by the KDC) for both conversations, then theoretically an intruder could swap the data from one conversation with the other, and confuse Alice and Bob. Alice could get two tickets for Bob, but instead, Kerberos allows Alice to choose a different key for a particular conversation and put that into the authenticator. If the authenticator has a session key that Alice inserted, Bob will use the Alice-Bob session key to decrypt the authenticator, but will use the session key Alice put into the authenticator in that conversation with Alice.

11.12 DOUBLE TGT AUTHENTICATION

Suppose Alice needs to access service Bob, but Bob does not know his master key. We'll assume Bob used his master key to obtain a TGT and session key, and then forgot his master key. Usually, if Alice asks for a ticket to Bob, the KDC will give her a ticket encrypted with Bob's master key. But Bob will not be able to decrypt the ticket, since Bob no longer knows his master key. If Bob is a user at a workstation, the workstation could at this point prompt Bob to type in his password again, but this would be inconvenient for the user.

Kerberos assumes Alice knows that Bob is the type of thing who is unlikely to know his own master key. In a method unspecified in Kerberos, Alice is supposed to ask Bob for his TGT. Alice then sends Bob's TGT as well as her own TGT to the KDC. (Hence the name **double TGT authentication**). Since Bob's TGT is encrypted under a key that is private to the KDC, the KDC can

decrypt it. It then issues a ticket to Bob for Alice which is encrypted with Bob's session key rather than Bob's master key.

The application which inspired this bit of the design was XWINDOWS. XWINDOWS clients and servers are backwards from what one might have guessed. The XWINDOWS server is the process that controls the user's screen. XWINDOWS clients are applications that make requests to the server to open windows and display information. While the user of an XWINDOWS terminal may need to authenticate himself to some remote application in order to start it, that application must authenticate itself to the XWINDOWS server to get permission to display its output. The human, Bob, logs into a workstation. The workstation then gets a TGT and session key on behalf of Bob and then promptly forgets Bob's master key. The application which is writing onto Bob's workstation must authenticate itself to the workstation. Since the workstation has no credentials other than Bob's TGT and session key, only a double TGT authentication as described above can work.

11.13 KDC DATABASE

Each entry in the V5 KDC database contains the following information. The structure of the database is somewhat implementation-specific, but since all current implementations are derived from the MIT implementation, we describe the MIT implementation.

- *name*—name of principal
- *key*—principal's master key
- *p_kvno*—principal's key version number. If this principal has *k* different valid keys, there will be *k* database entries for this principal. This could have been done more compactly by allowing multiple ⟨*key*, *p_kvno*, *k_kvno*⟩ entries per database.
- *max_life*—maximum lifetime for tickets issued to this principal
- *max_renewable_life*—maximum total lifetime for renewable tickets to this principal
- *k_kvno*—KDC key version under which *key* is encrypted
- *expiration*—time when this database entry expires
- *mod_date*—time of last modification to this entry
- *mod_name*—name of the principal who made the last modification to this entry

- flags indicating the KDC's policy on various things; for instance, whether to require pre-authentication data, whether to allow certain types of tickets such as forwardable, renewable, proxiable, postdated, and so on
- *password expiration*—time when password expires. This is used to force the user to change passwords occasionally.
- *last_pwd_change*—time when user last changed password
- *last_success*—time of last successful user login (i.e., last AS_REQ with correct preauthentication data)

11.14 KERBEROS V5 MESSAGES

Given that the Kerberos V5 messages are defined in ASN.1 notation, it isn't useful to show exact message formats. We will instead just list the information in each of the messages.

11.14.1 Authenticator

The authenticator is not a free-standing message, but rather is contained in a TGS_REQ or an AP_REQ. The entire thing is encrypted, using the key in the ticket that always accompanies an authenticator. Assume the authenticator is being sent in a message transmitted by Alice. When decrypted, the authenticator contains the following fields:

AUTHENTICATOR-VNO	version number (5)
CNAME, CREALM	Alice's name and realm
CKSUM	(optional) checksum of application data that might have been sent along with the AP_REQ
CTIME, CUSEC	time at Alice (in seconds, microseconds)
SUBKEY	(optional) key Alice would like to use instead of the key in the ticket, for the conversation with Bob
SEQ-NUMBER	initial sequence number that Alice will use in her KRB_SAFE and KRB_PROT messages to Bob
AUTHORIZATION-DATA	application-specific data limiting Alice's rights

11.14.2 Ticket

A ticket is not a free-standing message, but is rather carried in messages such as TGS_REQ, AS_REP, TGS_REP, AP_REQ, and KRB_CRED. A ticket given to Alice for use with Bob looks like this:

MSG-TYPE	message type (1)
TKT-VNO	protocol version number (5)
REALM, SNAME	Bob's name and realm
The remainder of the fields are encrypted with Bob's master key (unless this ticket was obtained using Bob's TGT as in §11.12 *Double TGT Authentication*).	
FLAGS	FORWARDABLE, FORWARDED, PROXIABLE, PROXY, MAY-POST-DATE, POSTDATED, INVALID, RENEWABLE, INITIAL (ticket was issued using AS_REQ rather than TGS_REQ) PRE-AUTHENT (user authenticated himself to the KDC before the ticket was issued) HW-AUTHENT (user was authenticated before ticket issued, using something like a smart card)
KEY	key to be used when communicating with Alice
CNAME, CREALM	Alice's name and realm
TRANSITED	Names of realms transited between Alice's realm and Bob's realm
AUTH-TIME, START-TIME, END-TIME, RENEW-TILL	timestamps. START-TIME and RENEW-TILL are optional. Described in §11.4 *Ticket Lifetimes*.
CADDR	(optional) the set of addresses from which this ticket will be valid
AUTHORIZATION-DATA	application-specific data limiting Alice's rights

11.14.3 AS_REQ

An AS_REQ is used to request a TGT. It can also be used to ask for regular tickets, but tickets requested with an AS_REQ (as opposed to a TGS_REQ) will return credentials encrypted with the requester's master key. The TGS_REQ contains a TGT, and the credentials returned in response to a TGS_REQ are encrypted according to the session key in the TGT. Let's assume that the request is on behalf of Alice in Wonderland. Let's assume she's asking for either a TGT or a ticket to Bob.

MSG-TYPE	message type (10)
PVNO	protocol version number (5)
PADATA	(optional) preauthentication data—timestamp encrypted with Alice's master key
KDC-OPTIONS	flags—each flag indicates a request to set the corresponding flag in the ticket the KDC will return (see below)
CNAME	Alice's name (the "c" comes from "client")
SNAME	Bob's name (or the name krbtgt if the request is for a TGT)
REALM	realm in which both Alice and Bob reside
FROM	(postdated ticket) desired start-time
TILL	desired end-time, which is the expiration time in the ticket
RTIME	desired renew-till time (only in request for renewable ticket)
NONCE	number to be returned in the reply to prevent replay attacks (MIT implementation uses current timestamp as the nonce)
ETYPE	type of encryption Alice would like KDC to use when encrypting the credentials
ADDRESSES	network layer addresses to include in ticket—used in proxy or forwardable tickets, or when Alice has multiple network layer addresses

The flags that make sense in an AS_REQ are:

- FORWARDABLE—Please set the FORWARDABLE flag in the returned TGT (so that the TGT can later be sent back to the KDC to request a TGT with a different network layer address inside).
- PROXIABLE—Please set the PROXIABLE flag in the returned TGT (so that the TGT can be used to request a ticket with a different network layer address inside).
- ALLOW-POSTDATE—Please set the ALLOW-POSTDATE flag in the returned TGT (so that this TGT can be used to request postdated tickets).
- POSTDATED—Make the returned ticket or TGT postdated, using the START-TIME in the request. Note that the START-TIME is an optional field, and it probably would have been more elegant to merely assume, if the requester included a START-TIME, that the requester wanted the ticket to be a postdated ticket. But the way Kerberos is defined, if the requester includes a START-TIME and does not set the postdated flag, then the START-TIME is ignored and an ordinary, nonpostdated ticket is returned.
- RENEWABLE—Please set the RENEWABLE flag in the returned ticket or TGT.
- RENEWABLE-OK—The requester wants a ticket with a long lifetime. If the KDC is not willing to issue a ticket with that long a lifetime, the requester is willing to settle for a renewable

ticket with an initial expiration time as far in the future as the KDC is willing to issue and renewable until the requested expiration time.

11.14.4 TGS_REQ

A TGS_REQ is used to request either a TGT or a ticket.

MSG-TYPE	message type (12)
PVNO	protocol version number (5)
PADATA	ticket and authenticator
KDC-OPTIONS	flags from AS_REQ, plus a few more explained above
SNAME	(or the name krbtgt if the request is for a TGT)
REALM	realm in which Bob resides (Alice might reside in a differerent realm in the case of a TGS_REQ)
FROM	(postdated ticket) desired start-time
TILL	desired end-time, which is the expiration time in the ticket
RTIME	desired renew-till time (only in request for renewable ticket)
NONCE	number to be returned in the reply to prevent replay attacks (MIT implementation uses current timestamp as the nonce)
ETYPE	type of encryption Alice would like KDC to use when encrypting the credentials
ADDRESSES	network layer addresses to include in ticket—used in proxy or forwardable tickets, or when Alice has multiple network layer addresses
AUTHORIZATION-DATA	application specific data to be copied into TGT and tickets requested using that TGT, intended to convey restrictions on use. Note that this field is encrypted and integrity-protected.
ADDITIONAL-TICKETS	Bob's TGT in the case where Bob does not know his master key (see §11.12 *Double TGT Authentication*)

The differences between a TGS_REQ and an AS_REQ are:

- The TGS_REQ contains a TGT or a renewable or postdated ticket (the AS_REQ does not).
- The TGS_REQ includes an authenticator in its PADATA field, proving the requester knows the key contained in the TGT or ticket in the request. The AS_REQ contains an encrypted timestamp in its optional PADATA field, proving the requester knows Alice's master key.
- The reply to a TGS_REQ is usually encrypted with the key inside the TGT or ticket enclosed with the request. However, if the authenticator contains a different key (called a

subkey), the reply is encrypted with the subkey inside the authenticator. In contrast, the reply to an AS_REQ is always encrypted with the requester's master key.

- There are more flags that might be relevant in a TGS_REQ. All the flags applicable to an AS_REQ are applicable to a TGS_REQ. In addition, the following flags are applicable in a TGS_REQ:
  - FORWARDED—A list of addresses appears in the request which is different than the list of addresses (if any) that appears in the ticket. The list in the request should be included in the returned ticket, and the FORWARDED flag should be set in the returned ticket.
  - PROXY—Same as FORWARDED, except this flag is used when requesting a TGT.
  - ENC-TKT-IN-SKEY—Included in this request is Bob's TGT (see §11.12 *Double TGT Authentication*).
  - RENEW—Please renew the enclosed ticket.
  - VALIDATE—Please validate the enclosed postdated ticket.
- The AS_REQ contains the field CNAME, which does not appear in a TGS_REQ. It is not needed in the TGS_REQ because the KDC obtains the name of the requester from inside the ticket or TGT enclosed with the TGS_REQ.
- The TGS_REQ contains the field AUTHORIZATION-DATA, and the AS_REQ does not. This field is supposed to be copied from the request into the ticket or TGT returned with the reply. It's actually somewhat of a nuisance that Kerberos does not allow this field in an AS_REQ. If you want a TGT or ticket with AUTHORIZATION-DATA, then you have to first obtain a TGT without that field, and then use that TGT in a TGS_REQ to request a TGT with AUTHORIZATION-DATA. Note that in order to prevent an intruder from modifying AUTHORIZATION-DATA in the request on its way to the KDC, the field is encrypted and integrity-protected with the key in the enclosed ticket or TGT, or if a subkey is present in the authenticator, then it's encrypted with that subkey. Note that AUTHORIZATION-DATA is treated differently than the other fields in the request, such as Bob's name, which are sent unencrypted and without integrity protection. Alice knows those other fields arrived intact because they are encrypted and integrity-protected when the KDC returns the credentials to Alice.
- The TGS_REQ also contains the field ADDITIONAL-TICKETS, which if ENC-TKT-IN-SKEY is set in the KDC-OPTIONS field in the TGS_REQ, contains Bob's TGT.

11.14.5 AS_REP

An AS_REP is the reply from the KDC to an AS_REQ. It returns a TGT or ticket.

MSG-TYPE	message type (11)
PVNO	protocol version number (5)
PADATA	(optional) salt to combine with the user's password in order to compute the master key derived from the user's password (see below)
CREALM	Alice's realm
CNAME	Alice's name. The purpose of Alice's name and realm is to help Alice's workstation figure out what key to use to decrypt the encrypted data.
TICKET	the ticket to Bob that Alice requested
ENC-PART	encrypted portion (see below)

In practice, PADATA is absent, indicating that the salt to be used is the user's name and realm. If a different salt is specified, it is not possible to transmit PADATA in the AS_REQ, because the user's master key would not be known.

Kerberos does provide mechanisms for recovery in case Alice's workstation does not know the proper value of salt. One plausible reason why Alice's workstation would not know the salt is that the realm name has changed since Alice last set her password. If the workstation has the wrong salt value, it will supply an incorrect value for PADATA in the request, and the KDC will return an error message. The error message returned by the KDC contains the proper salt value, and then Alice's workstation can try again, this time knowing the proper salt value.

The ENC-PART is encrypted with Alice's master key. When decrypted, it contains the following fields:

KEY	encryption key associated with the ticket enclosed in the AS_REP
LAST-REQ	a sequence of from 0 to 5 timestamps specifying such information as when Alice last requested a TGT, or last requested any ticket. The specification is vague about how these times are supposed to be synchronized across KDC replicas. Indeed, the MIT implementation (as of the writing of this book) does not implement any of these, and always returns no timestamps in this field.
NONCE	The nonce copied from the AS_REQ
KEY-EXPIRATION	(optional) time when user's master key will expire for the purpose of warning Alice to change her password
FLAGS	a copy of the flags that appear inside the ticket (so that Alice can check if the KDC granted all she requested in the request, and also allows her to detect malicious modification that might have been done to the AS_REQ)

AUTH-TIME, START-TIME, END-TIME, RENEW-TILL	timestamps; START-TIME and RENEW-TILL are optional (see §11.4 *Ticket Lifetimes*)
SREALM, SNAME	Bob's name and realm
CADDR	(optional) the set of addresses from which this ticket will be valid

11.14.6 TGS_REP

A TGS_REP is the reply from the KDC to a TGS_REQ. It is virtually identical to an AS_REP. The differences are:

- There is never a PADATA field in a TGS_REP, whereas it is optional in an AS_REP. (The PADATA field in an AS_REP contains the salt.)
- There is no KEY-EXPIRATION field in a TGS_REP, whereas it is optional in an AS_REP.
- The ENC-PART field is encrypted with the key in the TGT or ticket sent in the TGS_REQ; or if a subkey is included in the authenticator sent in the TGS_REQ, then the ENC-PART is encrypted with that subkey.

11.14.7 AP_REQ

An AP_REQ is the first message when Alice, who has obtained a ticket to Bob, actually attempts to communicate with Bob.

MSG-TYPE	message type (14)
PVNO	protocol version number (5)
AP-OPTIONS	flags, of which two are defined: USE-SESSION-KEY, which means the ticket is encrypted under the session key in Bob's TGT (rather than Bob's master key—see §11.12 *Double TGT Authentication*) MUTUAL-REQUIRED, which tells Bob mutual authentication is requested
TICKET	the ticket to Bob
AUTHENTICATOR	an authenticator, proving Alice knows the key inside the ticket

11.14.8 AP_REP

An AP_REP is Bob's reply to an AP_REQ from Alice.

MSG-TYPE	message type (15)
PVNO	protocol version number (5)
	the rest is encrypted:
CTIME	the time copied from the CTIME field of the authenticator in the AP_REQ
CUSEC	the low order bits of CTIME, since CTIME is expressed in seconds; this field specifies microseconds
SUBKEY	an optional field intended for Bob to be able to influence the Alice-Bob session key in an application-specific way
SEQ-NUMBER	starting sequence number for messages sent from Bob to Alice

The encrypted section (CTIME through SEQ-NUMBER) is encrypted with the key inside the ticket from the AP_REQ, unless a SUBKEY field is included in the AUTHENTICATOR from the AP_REQ, in which case it is encrypted with the subkey.

11.14.9 KRB_SAFE

A KRB_SAFE message transfers data between Alice and Bob with integrity protection.

MSG-TYPE	message type (20)
PVNO	protocol version number (5)
USER-DATA	whatever the application wants to send
TIMESTAMP	(optional) current time in seconds at the originator of the message, so the recipient can put messages in order, and can make sure the timestamp is within acceptable clock skew
USEC	(optional) the low-order bits (the microsecond portion) of the time, since TIMESTAMP is in seconds
SEQ-NUMBER	(optional) sequence number of this message, so the recipient can detect lost messages and put messages in order
S-ADDRESS	the network address of the sender of the message (the same address is presumably in the network layer header, but here it is cryptographically protected)
R-ADDRESS	the recipient's network address. Again, presumably it is equal to the destination address in the network layer header, but here it is cryptographically protected.
CKSUM	checksum on the fields USER-DATA through R-ADDRESS, using one of the checksum types defined in §11.8.1 *Integrity-Only Algorithms*

11.14.10 KRB_PRIV

A KRB_PRIV message is encrypted (and integrity-protected) data sent between Alice and Bob. It is encrypted with the key arranged for this conversation.

MSG-TYPE	message type (21)
PVNO	protocol version number (5)
	the rest is encrypted:
USER-DATA	whatever the applications wants to send
TIMESTAMP	(optional) current time in seconds at the originator of the message, so the recipient can put messages in order, and make sure the timestamp is within acceptable clock skew
USEC	(optional) the low order bits (the microsecond portion) of the time, since TIMESTAMP is in seconds
SEQ-NUMBER	(optional) sequence number of this message, so the recipient can detect lost messages and put messages in order
S-ADDRESS	the network address of the sender of the message (the same address is presumably in the network layer header, but here it is cryptographically protected)
R-ADDRESS	the recipient's network address. Again, presumably it is equal to the destination address in the network layer header, but here it is cryptographically protected.

11.14.11 KRB_CRED

A KRB_CRED message is used for passing credentials (a ticket and session key) for the purpose of delegation (see §11.3 *Delegation of Rights*). Assume Alice would like to delegate to Ted her right to access Bob. Alice would send Ted a KRB_CRED message containing a ticket to Bob, along with the session key corresponding to Bob's ticket. The encrypted portion of the KRB_CRED message is encrypted using a key that has been established between Alice and Ted, so the assumption is that Alice has already initiated a Kerberos protected conversation to Ted, and they now share a key.

MSG-TYPE	message type (22)
PVNO	protocol version number (5)
TICKETS	a sequence of tickets
the rest is encrypted with the Alice-Ted conversation key:	
TICKET-INFO	information corresponding to each ticket in TICKETS field, see below
NONCE	(optional) a number supplied by Carol to Alice, which Alice puts into the KRB_CRED message, when delegating to Carol, to reassure Carol that the KRB_CRED is not a replay transmitted by an intruder, and is indeed recently transmitted by Alice
TIMESTAMP	(optional) current time in seconds at the originator of the message, so the recipient can put messages in order, and make sure the timestamp is within acceptable clock skew
USEC	(optional) the low order bits (the microsecond portion) of the time, since TIMESTAMP is in seconds
S-ADDRESS	the network address of the sender of the message (the same address is presumably in the network layer header, but here it is cryptographically protected)
R-ADDRESS	the recipient's network address. Again, presumably it is equal to the destination address in the network layer header, but here it is cryptographically protected.

The TICKET-INFO field is a sequence of one or more repetitions of the following information:

KEY	encryption key associated with the corresponding ticket enclosed in the KRB_CRED
PREALM, PNAME	(optional) Alice's name and realm
FLAGS	(optional) a copy of the flags that appear inside the ticket
AUTH-TIME, START-TIME, END-TIME, RENEW-TILL	(optional) timestamps
SREALM, SNAME	(optional) Bob's name and realm
CADDR	(optional) the set of addresses from which this ticket will be valid

11.14.12 KRB_ERROR

In Kerberos V4 there were two types of error messages, one that would be returned by the KDC, the other returned by an application when authentication failed. In Kerberos V5 there is only one error message defined, and it is used for both purposes. None of the information in the error

message is encrypted or integrity-protected. Let's assume that Alice has sent a message to Bob, and that Bob is returning the error message to Alice because of some problem with Alice's message.

MSG-TYPE	message type (30)
PVNO	protocol version number (5)
CTIME, CUSEC	(optional) CTIME and CUSEC fields copied from the message generated by Alice that caused the error
STIME, SUSEC	time at Bob when he generated the KRB_ERROR message
ERROR-CODE	the error code, indicating the type of error
CNAME, CREALM	(optional) Alice's name and realm
REALM, SNAME	Bob's realm and name
E-TEXT	additional information to help explain the error, in printable text.
E-DATA	additional information to help explain the error. Not guaranteed to be printable text.

Here are all the error codes. Error codes 1–30 come only from the KDC, in response to a AS_REQ or TGS_REQ. The others can come from either the KDC or an application, in response to an AP_REQ, KRB_PRIV, KRB_SAFE, or KRB_CRED.

code	reason
0	no error. (Really, it's in the documentation! I'm sure it's annoying to get an error message telling you that you *didn't* make an error but it's not going to do what you asked it to do anyway. In reality, this would never appear, and is probably listed in the documentation just to ensure nobody assigns error code 0 to a real error.)
1	Alice's entry in the KDC database has expired.
2	Bob's entry in the KDC database has expired.
3	The requested Kerberos version number is not supported.
4	The KDC has forgotten the key with which Alice's entry in its database was encrypted. (It was an old version number, and the KDC didn't save that key.)
5	The KDC has forgotten the key with which Bob's entry in its database was encrypted.
6	The KDC never heard of Alice.
7	The KDC never heard of Bob.
8	Either Bob or Alice appears in the KDC database multiple times. (Really, it would make more sense to check this when modifying the KDC database, or have a utility that checks this every once in awhile rather than checking this when requests are made.)
9	Either Bob or Alice's entry in the KDC does not contain a master key (see parenthetical remark for error 8).
10	Alice asked for a postdated ticket, but her TGT does not allow this.
11	The requested start time is later than the end time (maybe the KDC should just give Alice the useless ticket that she requested).

code	reason
12	KDC policy does not allow the request.
13	KDC cannot grant the requested option.
14	KDC doesn't support this encryption type.
15	KDC doesn't support this checksum type.
16	KDC does not support this type of PADATA.
17	KDC does not support the transited type. (The TRANSITED field has a type and a value. The type is one that the KDC does not understand.)
18	Alice's credentials have been revoked—the account is marked invalid in KDC, or Alice's TGT has been revoked.
19	Bob's credentials have been revoked.
20	TGT has been revoked.
21	Alice's entry is not yet valid—try again later.
22	Bob's entry is not yet valid—try again later.
23	Alice's password has expired.
31	Integrity check on decrypted field failed.
32	The ticket has expired.
33	The ticket is not yet valid.
34	The request is a replay.
35	This ticket isn't for us.
36	The ticket and authenticator don't match.
37	The clock skew is too great.
38	The network address in the network layer header doesn't match the network layer address inside the ticket.
39	The protocol version number doesn't match.
40	The message type is unsupported.
41	The checksum didn't check. (Yhe documentation describes this error as *message stream modified*, but we prefer not to place blame—there might be a perfectly innocent explanation.)
42	The message is out of order. In both encrypted and integrity-checked data, there is a sequence number. The network can certainly reorder messages, and such innocent reordering of messages should not generate an error. If the messages are being delivered with a reliable transport layer protocol, then this error would be generated because of some deliberate tampering with the message stream, which can be detected by Kerberos. If the messages are being delivered with a datagram service (like UDP), then since the sequence number is optional, it should not be used.
44	The specified version of the key is not available.
45	Bob doesn't know his key.

code	reason
46	Mutual authentication failed.
47	The message direction is incorrect. In V4 this was determined based on the D BIT. The D BIT no longer exists in V5, but instead, in integrity-protected and encrypted data, there is a SENDER'S ADDRESS and a RECEIVER'S ADDRESS field. If Bob receives a message from Alice and these fields are swapped, it indicates that an intruder is trying to trick them by mirroring messages back.
48	*alternative authentication method required.* For instance, Bob does not know his master key, but does have a TGT (see §11.12 *Double TGT Authentication*).
49	The sequence number in the message is incorrect.
50	*inappropriate type of checksum in message.* Presumably, this is because Bob doesn't support that checksum type. The wording in the documentation implies that Bob is making a value judgement on Alice's choice of checksum. For instance, if Alice were to choose CRC-32 as an integrity check instead of a message digest, Bob might sneeringly send this error message.
60	generic error. The description is in E-TEXT.
61	The field is too long for this implementation.

11.15 HOMEWORK

1. Suppose the Kerberos V5 password to key conversion function is identical to V4, but then takes the output that V4 would compute and ⊕s it with the realm name. This would produce a different key in each realm, as desired. What is wrong with this algorithm? (Hint: the reason it is good for your key to be different in different realms is so that if your key in one realm is known, it does not divulge your key in other realms.)

2. Consider the following variant of Kerberos. Instead of having postdated or renewable tickets, a server which notes that the start-time is older than some limit presents the ticket to the TGS and asks if it should believe the ticket. What are the tradeoffs of this approach relative to the Kerberos V5 approach?

3. The philosophy behind requiring renewable tickets to be renewed before they expire is that a KDC should not need to remember blacklist information indefinitely. But does that work for postdated tickets, given that a postdated ticket can be requested with a start-time arbitrarily far into the future?

4. Design a different method of Bob authenticating Alice when Bob does not remember his own master key, which places the work on Bob instead of Alice. In other words, Alice will act as

if Bob was an ordinary civilized thing that does remember its own master key, and Bob interacts appropriately with the KDC so that Alice will be unaware that Bob didn't know his own master key.

5. Suppose it was desired to use Kerberos for securing electronic mail. The obvious way of accomplishing this is for Alice, when sending a message to Bob, to obtain a ticket for Bob and include that in the email message, and encrypt and/or integrity-protect the email message using the key in the ticket. The problem with this is that then the KDC would give Alice a quantity encrypted with Bob's password-derived master key, and then Alice could do off-line password guessing. How might Kerberos be extended to email without allowing off-line password guessing? (Hint: issue human users an extra, unguessable master key for use with mail, and extend the Kerberos protocol to allow Bob to safely obtain his unguessable master key from the KDC.)

6. In the mutual authentication in the Needham/Schroeder protocol upon which Kerberos is based, the authenticator contained only an encrypted timestamp. The protocol is that Alice sends Bob the authenticator, and then Bob must decrypt the authenticator, add one to the value inside, re-encrypt it, and send it back to Alice. Why was it necessary for Bob to increment the value before re-encrypting it and sending it to Alice? Why isn't it necessary in Kerberos V5, in the AP_REP message? In Kerberos V4, it is the checksum field (which isn't really a checksum—see §10.10 *Encryption for Integrity Only*) that is extracted and incremented. Would it have been just as secure in V4 for Bob to send back the contents of the checksum field encrypted and not incremented?

7. In the KRB_SAFE message, there is both a timestamp and a sequence number. Presuming that both timestamp fields (TIMESTAMP and USEC) are sent, and that the application makes sure the timestamp increases on every message, does the sequence number provide any additional protection?

8. Prove that if there is a 64-bit value that works as a DES checksum, the value $\oplus$'d with $\text{F0F0F0F0F0F0F0F0}_{16}$ will also have a correct DES checksum.

9. Give the algorithm for verifying the MIC described in §11.8.1.2 *des-mac*.