Cryptanalysis Techniques: An Example Using Kerberos

Jennifer Kay

September 1995
CMU-CS-95-115

School of Computer Science
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213-3890

This is a revised version of a report that first appeared in February 1995.

Abstract

This paper is designed to be a guide to techniques for analyzing a cryptosystem. The Kerberos authentication system is used as an example, and the basis for the analysis is a working draft of the system: version 5 revision 5.1. More recent versions of the system have fixed many of the problems described here.

The Kerberos authentication system uses a trusted key server to keep track of the private keys of clients and servers, as well as to generate session keys for client-server interaction. If a client were able to determine the private key of a particular server, it could decrypt conversations that other clients have with that server, as well as impersonate other clients to that server.

Part of the Kerberos method of client-server conversation is the Kerberos "ticket", a record that is partially encrypted in the private key of the server. The ticket contains the session key, information about the client, and some other useful data. Since the ticket's format is widely available, and most of the contents of the ticket are known to the client, it is potentially vulnerable to chosen or known plaintext attacks.

Jennifer Kay is supported by a National Aeronautics and Space Administration Graduate Student Research Fellowship. Support has come from "Perception for Outdoor Navigation", contract number DACA76-89-C-0014, monitored by the US Army Topographic Engineering Center, and "Unmanned Ground Vehicle System", contract number DAAE07-90-C-R059, monitored by TACOM.

Keywords: Cryptanalysis, Security and Protection, Authentication, Kerberos

1. Introduction

This paper is designed to be a guide to the sorts of techniques used for analyzing a cryptosystem. The Kerberos authentication system is used as an example, and the basis for the analysis is a working draft of the system: version 5 revision 5.1. More recent versions of the system have fixed many of the problems described here.

The Kerberos authentication system uses a trusted key server to keep track of the private keys of clients and servers, as well as to generate session keys for client-server interaction. If a client were able to determine the private key of a particular server, it could decrypt conversations that other clients have with that server, as well as impersonate other clients to that server.

I begin in section 2 by describing in more detail the structure of the Kerberos method of authentication. The working draft that I analyzed allowed users of Kerberos to choose the private key encryption method that the system would use, and the rest of the paper describes the relationship between the type of cryptosystem used and possible attacks on that system.

Part of the Kerberos method of client-server conversation is the Kerberos “ticket”, a record that is partially encrypted in the private key of the server. Since the ticket’s format is widely available, and most of the contents of the ticket are known to the client, it is potentially vulnerable to a chosen or known plaintext attack. In section 3, I discuss several avenues of attack on the system using multiple submission of ticket requests. Finally, in section 4, I review the Data Encryption Standard (DES) method of encryption in particular, and look at how the method of differential cryptanalysis could be used for an attack against a Kerberos system using DES.

2. Kerberos Tickets

2.1. Overview of tickets and the possibility of a chosen plaintext attack.

The procedure used by a client to get a service in Kerberos is well documented, and so I shall limit my description of this procedure to a very high level. An excellent description of Kerberos can be found in [Steiner].

An authenticated client may request a “ticket,” to be used in communication with a particular server from a special server known as the “ticket-granting server” (TGS). The TGS returns to the client the ticket and session key to be used when communicating with the server. The ticket, along with a client-generated “authenticator,” is sent to the server each time the client wants to use a service.

In order to communicate with the TGS, the client must initially request a special ticket from the “authentication server.” The request for this special ticket contains the identity of the client and the TGS it wishes to use. The authentication server passes back to the client the session key which the client should use to communicate with the TGS, encrypted with the client’s private key, which is derived from their password, along with a ticket to give to the TGS when requesting other tickets. This ticket is known as the “ticket-granting ticket.”

There are 3 attributes of the Kerberos system of acquiring tickets from the TGS that might enable the use of tickets to discover a server’s private key: parts of the ticket are encrypted using the server’s private key, portions of the encrypted part of the ticket can be completely specified by the client in its request to the TGS, and there does not appear to be any restriction on a client against asking the TGS for multiple tickets for the same server. Thus the possibility for a chosen plaintext attack is evident. We can repeatedly ask for tickets from the TGS, changing only our area of interest and use the plaintext/ciphertext pairs to try and compute the server’s private key.

2.2. The Encrypted Part of a Kerberos Ticket.

The following details are based on a working draft of the Kerberos version 5 system[Kerb10-92] which are very similar to the protocol presented in a draft written 3 years previously[Kerb11-89].

A Kerberos ticket has a cleartext part and an encrypted part. As was noted earlier, the encrypted part is encrypted with the private key of the server. There are several different protocols for encryption that are defined in the Kerberos document, and I shall look at the encryption layer later on in section 3.

Following is a description of each of the fields of the encrypted part of a ticket, along with what the client knows about this field. A field labelled optional is not required to be included in a ticket:

flags: a bit-field indicating certain options used in the ticket.

- When requesting a ticket the client can specify all of the bits in this field.

key: the session key to be used for client-server communication.

- The TGS returns this key to the client when it sends the client the ticket
- This is the only field (other than those specific to the encryption layer) which could change with repeated identical requests to the TGS for a ticket for the same server.

crealm: the name of the client’s current realm.

- This is known to the client.

cname: the clients name.

- This is known to the client.

transited: if the user needed to cross realms to get this ticket, this field specifies all the realms involved.

- For simplicity, we will assume that the client is trying to find the key of a server in the same realm, and thus this field contains the empty string.

auth_time: the time the client was initially authenticated.

- This is set by the authentication server when it creates the ticket-granting ticket. This is known to the client.

starttime: (optional) the time at which the ticket starts to be valid.

- The client may specify this field in the ticket request.

endtime: the time at which the ticket expires.

- The client may specify this field in the ticket request. The client should be sure to make

$$endtime - starttime < MaxLenOfTicket$$

renew-till: (optional) if the ticket is renewable (as indicated in the flags field), then this field contains the latest possible time this ticket is valid even if renewed.

- The client may specify this field in the ticket request.

caddr: (optional) the address from which this ticket may be used.

- The client may specify this field in the ticket request.

authorization data: (optional) this field can contain any data that the client wishes to pass on to the server.

- The type of data is intended to be dependent on the server that a client is using, but there is no real restriction on what this field contains. Thus the client may insert any data it chooses in this field.

3. Encryption

3.1. Introduction.

Kerberos does not require that one specific method of encryption be used. Instead, the designers of Kerberos have defined multiple encryption protocols, and presumably would be willing to add others to the list should they seem promising.

In this section I will consider how tickets are encrypted, and look at some properties of tickets and forms of encryption that may or may not make it more difficult to find a server's key. All of the protocols currently defined for Kerberos use ciphers that encrypt fixed sized blocks of code. The phrase "encryption method" is used to refer to the method used to convert a plaintext block into an encrypted block, and the phrase "chaining method" is used to refer to any techniques used to randomize the plaintext block before this encryption.

3.2. High Level Encryption Specifications.

Although Kerberos is flexible in the method of encryption that is used in a particular system, the format of an encrypted message is the same regardless of the actual encryption protocol. It is this format that is presented in this section.

Before encrypting a message in Kerberos, two optional fields may be added to the head of the message. The length of each field is determined by the encryption method. The first field is known as the “confounder.” The confounder is a random byte string which has been inserted in the ticket by the TGS, and is unknown to the client.

The second optional field is used to store a checksum. No particular method of checksum is required, but the checksum is computed by zeroing out the checksum field and computing the appropriate checksum based on the whole message (including the confounder part). The resulting value is inserted in the checksum field.

The confounder is intended to make chosen and known plaintext attacks more difficult by adding unknown data to the head of the message. The checksum is used to detect any tampering with a ticket that might have occurred after the ticket was issued.

3.3. Electronic Code Book Algorithms: Completely Unconfounding.

If the confounder does not have influence on the encrypted value of all of the successive blocks of the message then it is essentially useless. Consider an encryption algorithm that works on blocks of data, each block being of length n. In an “electronic code book” style method of encryption[Davies], each block of the plaintext is encrypted individually, so that the ciphertext is completely dependent on the individual block, and independent of any other block. In terms of the phrases introduced in section 3.1., the NULL “chaining method” is used here, i.e. the plaintext is not changed before encryption.

If we choose an encryption scheme for Kerberos that uses electronic code book style encryption, then the choice of confounder has no influence on the “authorization-data” field of the ticket. If we set up our authorization data field so that we pad out the current data block as necessary, and then put a particular chosen plaintext in the next whole block, we can determine the encrypted value of that chosen plaintext. Thus any method of encryption that is breakable by using a chosen plaintext attack is breakable in Kerberos with electronic code book style encryption.

3.4. Cipher Block Chaining Algorithms: Possibly Confounding.

From the above discussion it is obvious that we need some form of encipherment where the encrypted version of a given block is dependent on the preceding blocks. All of the encryption protocols currently defined in Kerberos (with the exception of the “NULL” protocol which sets the ciphertext equal to the plaintext) use the “cipher block chaining” method to do this. Cipher block chaining is a “chaining method” in which the current plaintext is randomized by performing a “bitwise exclusive or” between it and the encrypted value of the previous block[Davies]. Then this randomized data is encrypted using the “encryption method” of your choice. There is a special “initialization vector” used for the first block of plaintext.

With cipher block chaining, every block of the data is dependent on all of the previous blocks of data. So the confounder has the possibility of being confounding. In the following sections I consider how confounding it actually is.

3.5. Confounders, Cipher Block Chaining, and Known Plaintext Attacks.

Confounders don't confound much against known plaintext attacks. A known plaintext attack, as opposed to a chosen plaintext attack, is one in which the person trying to find the encryption key is not allowed to choose the plaintext that is encrypted, but is given a copy of some plaintext and the associated ciphertext.

Consider an attempt to discover a particular server's key, and assume that all messages contain both a confounder and a checksum, and that cipher block chaining is used with the chosen encryption scheme. It may not be possible to determine the plaintext value of the first block of the data, that is, the part that contains the confounder and the checksum. However, move along a few blocks. As was shown in section 2.2., we know the plaintext value, before "randomization" using the chaining method, of most of the data.

Let us again use a piece of the "authorization-data" field that lines up on a block boundary. We know the plaintext before randomization of this field, because we wrote it. Once we have the ticket, we also know the encrypted value of the previous block, and of this one. If we do a bitwise exclusive or between our plaintext block, and the previous block's encrypted value, we have the exact "plaintext" that was encrypted to give us the encrypted value of the current block.

We have as many known plaintext-ciphertext pairs as we desire, and the confounder need not disturb us at all.

3.6. How Random Is Random?

The confounder and the session key are both chosen using a pseudo-random number generator (PRNG). If we can "break" the PRNG, and thus know the next in the sequence of "random" numbers, then we will know the values of the confounder and the session key before we request the next ticket. In later sections I show how this can be used to our advantage.

One very popular method of generating pseudo-random numbers is the "linear congruential method." [Knuth]. In this method, the sequence of random numbers $\langle X_n \rangle$ is generated by using the following equation:

$$X_{n+1} = (aX_n + c) \bmod m \tag{1}$$

Where:

$$\begin{gathered} m > 0 \\ 0 \le a < m \\ 0 \le c < m \\ 0 \le X_0 < m \end{gathered} \tag{2}$$

Although this type of a sequence is not “random” for all choices of variables, appropriate numbers can be chosen to make the sequence appear more random. Sometimes, rather than reporting all of x_i, only part of it is given, for example, the first k bits.

This type of a PRNG is not impossible to break. In other words, it is possible to completely predict subsequent values produced by the PRNG after seeing a sample of its output. Plumstead showed that if all of x_i is reported by the PRNG (as opposed to only a portion of it) then without knowing any of the variables in (2) it is still completely predictable. Frieze, Kannan, and Lagarias proved that if m and a are known then the sequence is completely predictable after seeing only a few of the x_i’s. Stern has proven it is possible to predict the sequence with a high probability knowing only m, but after viewing more of the sequence [Stern]. Finally, Frieze, Hasted, Kannan, Lagarias, and Shamir provided a polynomial time algorithm that can correctly predict the bits of “random” numbers generated using linear congruential generators [Frieze].

In the current working drafts of Kerberos V. 5 it is unclear what type of a PRNG is being used, however it is quite possible that it will be some type that is breakable. This provides a serious threat as will be shown in the next section.

3.7. The Confounder as a Known Plaintext.

I have shown that it is quite possible that the PRNG can be broken in such a manner that every subsequent “random” number it generates can be predicted. If this is the case, and if the confounder is at least as big as the blocksize, then we can use it as known plaintext. We know the value of the confounder, because we have broken the PRNG. We know the encrypted value of the confounder, because it is given to us by the TGS in the ticket. Thus the confounder can be used to help an evil adversary in an attack.

3.8. The Birthday Paradox.

The general form of the “birthday paradox” is as follows: how many people need to be in a room before the probability that two of them were born on the same day of the year is sufficiently high [Cormen]. Assume that birthdays are probabilistically independent.

If p were the probability that out of k people no two have the same birthday, then

$$1-p \tag{3}$$

would be the probability that at least one pair share a birthday. Assume that we know the probability, q, that out of k-1 people, no two share the same birthday. Then if n=365 (ignoring the annoying problem of leap years), we have:

$$p = \frac{n-(k-1)}{n} \times q \tag{4}$$

Obviously, the initial condition, k=1, gives p=1, i.e. if we have only one person in the room, the probability that he has a distinct birthday from everyone else in the room is 1.

Now let us consider this in the context of a PRNG. Let us assume that the PRNG is good, that is, that the probability of the next number generated is pretty much independent of the numbers already generated. Then the probability that we will have a duplicate number produced is given by (3) when n in (4) is set to be the number of distinct values this PRNG can produce. If n is fairly small, say 2^{16} (the size of the confounder [Kerb 11-89]), then the probability that we will have a matching pair of numbers from the PRNG is greater than 0.5 after about 300 numbers have been generated.

3.9. The Confounder as our Chosen Plaintext.

The title of this section is intriguing. By definition the confounder is supposed to confound us. We are not supposed to know its value, let alone choose it's value. In some sense, however, we may in fact be able to choose it's value.

Assume that we have broken the PRNG, and that the confounder and our blocksize are the same small size. Thus, for any given ticket, before we request the ticket we know what the confounder will be. Also assume that there is some block after the confounder block that we can choose the plaintext value of (call this B), and that all the fields between the confounder and our chosen field remain the same for each ticket. Choose a plaintext value that you want to know the encrypted value of.

Keep track of the following information for several tickets: the confounder value, and the encrypted value of the block immediately preceding your chosen plaintext block (call this value E). When you predict that the next confounder is going to be one of the ones you have seen before, xor your chosen plaintext with E, and insert this xor value into your ticket request as your value for B. When the CBC encryption is done, B will be xor'ed with the E again, and thus your chosen text will be the value encrypted. By the birthday paradox, since the confounder is small it shouldn't take too long before we have a duplicate entry which allows us to encrypt our chosen plaintext.

3.10. The Checksum: Just Another Confounder?

The purpose of the checksum in an encrypted Kerberos message is not so much to confound, as to prevent some party from tampering with the ticket. However, since it adds possibly unknown data to the message, it may be viewed as an additional confounder. Is this really the case?

Suppose that we have broken the PRNG, and therefore before we actually request the ticket, we know the value that will be placed in the confounder field. Thus, we can compute the value of the checksum. Depending on the complexity of the checksum algorithm, we may or may not be able to extend the authorization_data field in such a way as to "choose" the value of the checksum. If we can "choose" the checksum, it seems less confounding than the confounder. However, if we can not do so, it's contents may be even more confounding than the confounder itself.

4. The Data Encryption Standard

4.1. Introduction.

Other than the "NULL" encryption protocol, all of the Kerberos encryption protocols use the Data Encryption Standard (DES) for their method of encryption. DES is a non-linear method of encrypting eight byte blocks of data which was adopted by the National Institute of Standards and Technology as the algorithm to be used for protection of computer data. It was initially adopted as the standard for a five year period, was renewed twice, and is currently up for renewal again[Fed Reg]. In all likelihood DES will be renewed yet again, but this will probably be the last time. [Branstad]

DES has a key size of 56 bits, and thus an exhaustive search of the key space would have a complexity of 2^{56}. Until recently, no attack on DES has been shown to be faster than half of that, or 2^{55}. [Biham 90] However, in 1991 Biham and Shamir published an attack against DES [Biham 91] which uses 2^{37} time and analyzes 2^{36} ciphertexts using their differential cryptanalysis technique. In the following sections I review this technique, and look at it's applicability to Kerberos.

4.2. How DES Works.

Excellent descriptions of DES are available[Davies] and I will not attempt to cover DES to the same granularity that they achieve, however this section should give the reader unfamiliar with DES a very basic understanding of the system.

DES is an iterated cryptosystem, that is, the same procedure is repeated several times in a row for greater confusion. In the DES case, the data undergoes 16 rounds of the repeated function.

The DES key is 56 bits plus 8 checkbits (completely dependent on the 56 bits). Each round of the function, uses a 48 bit subkey, derived from the key, but different for each round. DES works on blocks 8 bytes (64 bits) in length.

The input to each round in DES is an 8 byte quantity, initially the plaintext block to be encrypted. In any given round, call this round r, first the right 32 bits of the block are encrypted. This encryption process will be described in a moment. The first 32 bits of output of round r of DES (i.e. the left half), is just the right half of the input to round r. The right half of the output is the result of the encryption process mentioned above, xored with the left half of the input to round r. This is summarized in Figure 1.

Now all that remains to be explained is the encryption function used on the right half of the data. The right half of the data is first expanded from 32 bits to 48 bits using the "expansion permutation" that duplicates 16 particular bits of the 32 bits and then permutes the resulting 48 bit data using a fixed permutation algorithm. The permuted 48 bits are xored with the 48 bit subkey for the current round. Then the 48 bits are divided into 8 groups of 6 consecutive

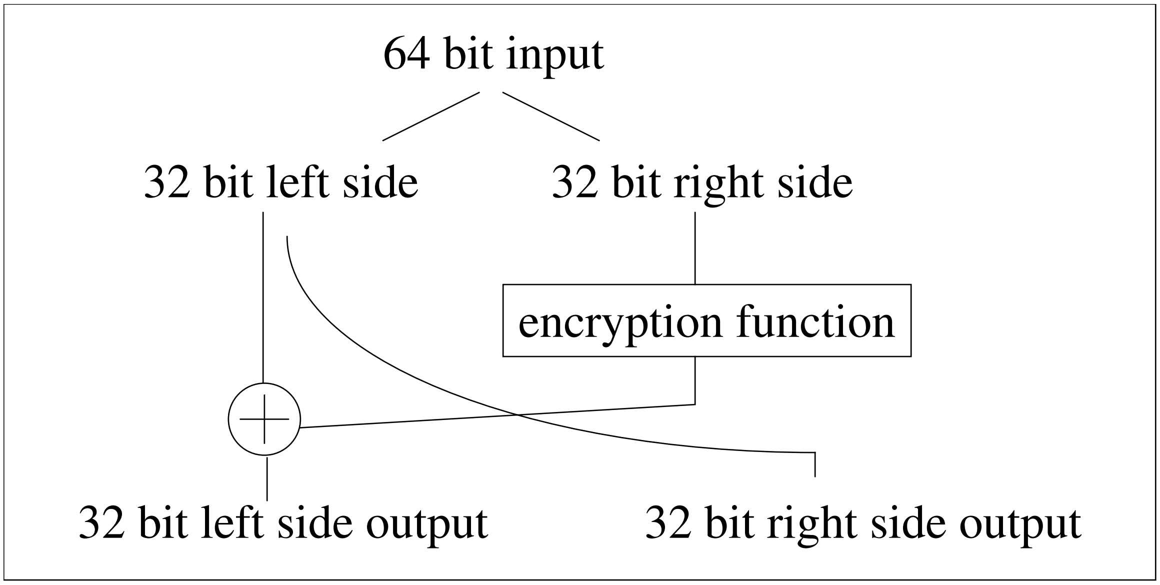

Figure 1: A single round in DES

bits each. Each 6 bit quantity is sent through an “S-box”, which can be thought of as a look-up table that turns a 6 bit quantity into a 4 bit quantity. There are 8 different look-up tables that correspond to the 8 different S-boxes in the encryption function. The concatenated 32 bit output from the S-boxes is again permuted, and this permuted 32 bit quantity is the output from the encryption function. This function is summarized in Figure 2.

4.3. Differential Cryptanalysis.

Differential cryptanalysis is a probabilistic attack on DES which attempts to find the most likely key based on the data seen. The method uses pairs of plaintext and their corresponding pairs of ciphertext. The plaintext pairs are chosen on the basis of their bitwise xor value.

Motivation for the method can be seen by looking at a small portion of the encryption function, specifically at one of the S-boxes. Assume that you know the input xor of a particular pair of 6 bit quantities just before they enter the same S-box. Note that this is the same as the xor of that pair of 6 bit quantities immediately before they have each been xored with the 6 bit portion of the subkey, since the subkey’s effect in each cancels itself out. For any given xor, there are $64 = 2^6$ ordered pairs of 6 bit quantities that have that same input xor. If you run each of these pairs (individually) through the S-box, you can compute the xor value of that particular output pair. You can imagine creating a table for each S-box. The rows of the table correspond to the input xor values (labelled from 0 to 2^6) and the columns correspond to the output xor values (labelled from 0 to 2^4). The value of a particular <row, column> entry is the number of xor pairs whose input xor to the S-box is <row> and whose output xor from that S-box is <col>. From this table, we can deduce the probability that a given input xor to an S-box would produce a particular output xor. More details on this can be found in [Biham 90].

Just as we can compute a probability for an <input, output> xor pair to a single S-box, we

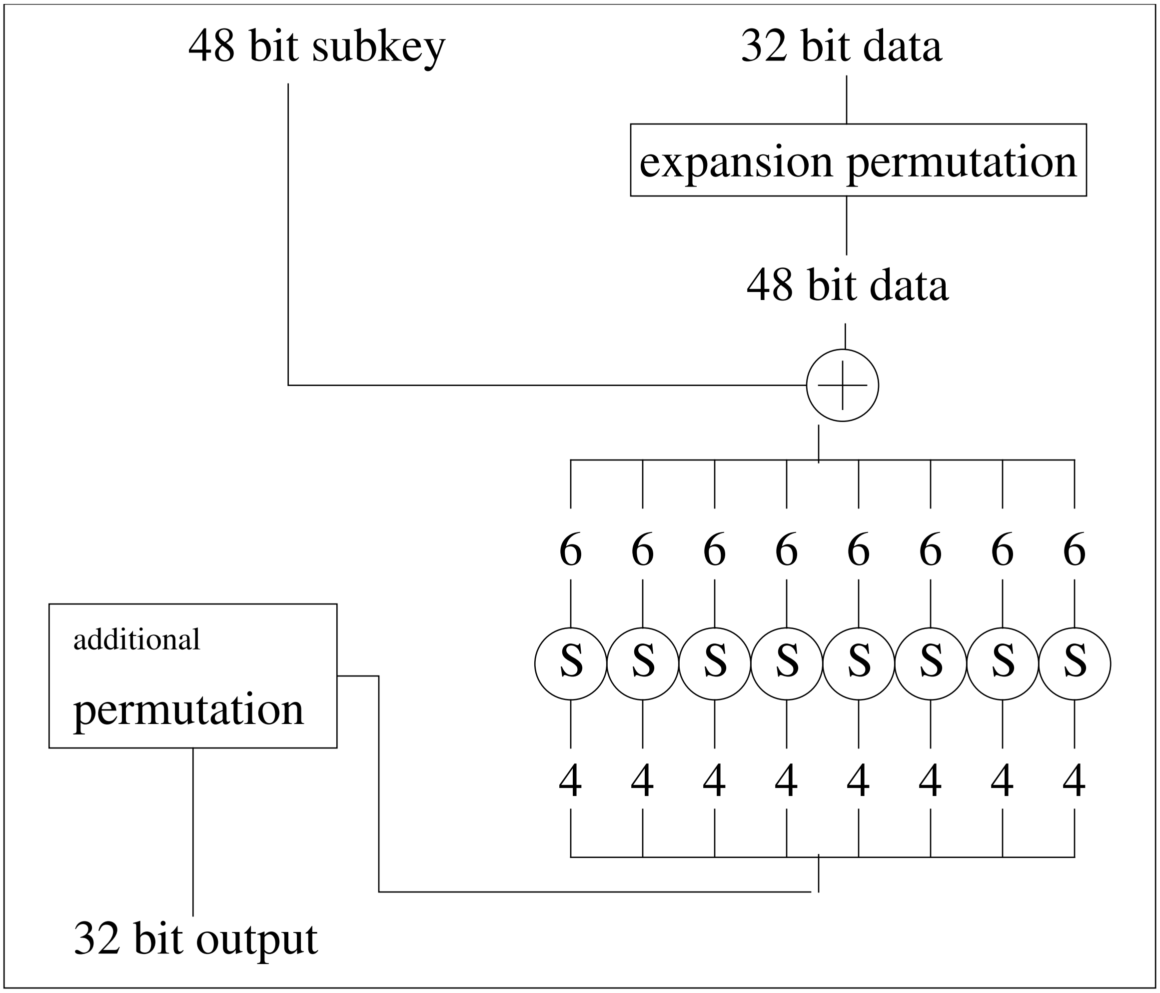

Figure 2: The Encryption Function

can compute a probability for a 64 bit <input, output> xor pair to a given round of DES. Further, we can combine such probabilities to compute the probability of a 64 bit <input, output> pair to the full 16 round version of DES. If we test enough data samples, we can determine which key is most likely based on probabilistic reasoning. In their most recent paper [Biham 91], Biham and Shamir show how they can use these techniques with a large set of chosen plaintexts to find the key with a very high probability.

4.4. Differential Cryptanalysis and Breaking Kerberos.

Biham and Shamir’s attack on DES is of the “chosen plaintext” variety. Thus the cipher block chaining of the current working draft’s recommendation effectively prevent it’s use in breaking Kerberos. It is important that the cipher block chaining remain a part of all encryption methods that use DES to prevent Biham and Shamir’s method to be usable.

Under some circumstances, even with cipher block chaining, Biham and Shamir’s method would be applicable. The reasoning is as follows: suppose that you know the encrypted value of the block immediately preceding your first full block of your authorization data field *before* you request the ticket. This is not impossible, and I shall soon discuss ways that this could happen. Anyway, if we knew the encrypted version of that block, then we could choose the plaintext version of our full authorization data field block so that when it is xored with the previous encrypted block the result is the "plaintext" value that we actually wish to encrypt. We're back to having our chosen plaintext encrypted and Biham and Shamir's method may proceed.

Under what circumstances would we know the encrypted value of the block immediately preceding the first full block of our authorization data field? It would be true in the case where there were no confounder, checksum, and if the key field containing the session key were relocated after the authorization data field. In this case, if you made intelligent ticket requests, none of the data preceding the authorization data field would ever change, and thus having received one ticket you would know the constant value for that encrypted block.

Suppose the size of a key is fairly small, and so by the birthday paradox you don't have to wait very long to "randomly" choose the same value for the key again. Then even if the key field is before the authorization data field, we could encrypt our text message several times until the encrypted value of the block containing the key is the same as our first attempt, and then look at the resulting ciphertext of our authorization data field block. Similarly we could wait for a small confounder to reappear, and apply the method.

Obviously, the above argument is equally applicable to any method of encryption that can be broken using a chosen plaintext attack.

5. Conclusions

5.1. Other Encryption Schemes

It could certainly be argued that the trusted key server concept is unsafe. Why not use public key encryption for everything? Consider the following system:

Every client and server has a unique public key. Anyone can encrypt a message using this key, but only the owner of the key can decrypt the message again. We also have the property that if you do the decryption procedure on a plaintext message, that the associated public key encryption procedure will produce the original message again.

In this scenario, we have one special server (SS), and we make sure that every other client and server knows SS's public key. In addition, SS knows everyone else's public key. Every client or server, when it creates or changes its public key, asks the SS for a certificate. The plaintext version of this certificate is the following: "I am SS and I certify that this is <client or server's name> and that it's public key is <it's public key>." The actual certificate that is given to the client or server is this plaintext certificate "decrypted" with the SS's secret key. Thus anyone can encrypt the certificate using the SS's public key to read the plaintext version, and know that only the SS could have generated that message.

Let's say that machine "A" wants to talk securely to machine "B", but neither know the other's public key. "A" can send a plaintext message to "B": "I am A, here is my certificate, could I please have yours." "B" can reply with his certificate. Each of them can encrypt the certificates using SS's public key. At this point, A knows B's public key, and knows that it is genuine, and vise-versa. So now they can carry on a private conversation, secure in the knowledge that only the other can decrypt messages sent to them.

One might object that we are using the SS as a trusted key server. If anyone discovered the way to decrypt SS's messages, anyone could fake them. This is true, however in our scenario it is less likely that a malicious client could discover the secret key: whereas in Kerberos the TGS has to be on the network, and is constantly sending out tickets, our SS can be quite isolated electronically. Further, if the TGS goes down, no new session keys can be issued, and two perfectly healthy machines may not be able to communicate. In the public key method, the SS going down has no effect on whether two machines with certificates can communicate.

5.2. Kerberos is Improving.

Several of the deficiencies of version 4 of Kerberos have been fixed in the proposed protocol for version 5. Even the early version 5 propositions had some problems that have been corrected in the current working draft, for example the confounder field in the older version 5 proposal was only two bytes long, and there was no checksum proposed at all, thus creating a confounder that doesn't confound.

The Kerberos designers must be careful to keep their system safe from the sorts of attacks against tickets that are described above. It would be fairly simple for the TGS to keep a record of <client, server> pairs for which the client requested a ticket for the given server in the last small period of time, and to delay replies to clients requesting tickets who request too many for the same server within a short time. This would aid in preventing a malicious client from gathering too much information about a server in any reasonable amount of time.

Key servers whose service should be kept confidential also need to change their private keys frequently enough to avoid these attacks. If a server's private key is only changed once a year, the possibility of discovering that key is much increased.

6. Acknowledgments

Many thanks to Doug Tygar for all his help and guidance in this work.

7. References

[Biham 90] E. Biham and A. Shamir, *Differential Cryptanalysis of DES-like Cryptosystems*, Technical Report CS90-16, Department of Applied Mathematics and Computer Science, The Weizmann Institute of Science, July 1990.

[Biham 91] E. Biham and A. Shamir, *Differential Cryptanalysis of the Full 16-round DES*, Technical Report 708, Department of Computer Science, Technion - Israel Institute of Technology, December 1991.

[Branstad] Dennis K. Branstad, National Institute of Standards and Technology, Personal Communication, 1992.

[Cormen] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, *Introduction to Algorithms*, MIT Press, McGraw-Hill Book Company, 1990.

[Davies] D.W. Davies and W.L. Price, *Security for Computer Networks*, John Wiley & Sons, 1989.

[Fed Reg] National Institute of Standards and Technology, "Proposed Reaffirmation of Federal Information Processing Standard (FIPS) 46-1, Data Encryption Standard (DES)", *Federal Register*, Vol. 57, No. 177, September 11, 1992.

[Frieze] Alan M. Frieze, Johan Hastad, Ravi Kannan, Jeffrey C. Lagarias, and Adi Shamir, "Reconstructing Truncated Integer Variables Satisfying Linear Congruences", *SIAM J. Computing*, vol. 17, no. 2, April 1988.

[Kerb 11-89] John Kohl, B. Clifford Neuman, and Jennifer Steiner, *The Kerberos Network Authentication Service*, MIT Project Athena, Version 5, Draft 2, 6 November 1989.

[Kerb 10-92] John Kohl and B. Clifford Neuman, *The Kerberos Network Authentication Service(V5)*, Internet Draft, Version 5, Revision 5.1, 1 September 1992.

[Steiner] Jennifer G. Steiner, Clifford Neuman, and Jeffrey I. Schiller, "Kerberos: An Authentication Service for Open Network Systems", *Proceedings of the USENIX Winter Conference*, 1988.

[Stern] Jacques Stern, "Secret Linear Congruential Generators are not Cryptographically Secure", *Proceedings of the IEEE Symposium on Foundations of Computer Science*, 1987.