On Strengthening Authentication Protocols to Foil Cryptanalysis*

Wenbo Mao and Colin Boyd

Communications Research Laboratory
Department of Electrical Engineering
University of Manchester
Manchester M13 9PL, UK
Email: wenbo@comms.ee.man.ac.uk

Abstract. Cryptographic protocols have usually been designed at an abstract level without concern for the cryptographic algorithms used in implementation. In this paper it is shown that the abstract protocol definition can have an important effect on the ability of an attacker to mount a successful attack on an implementation. In particular, it will be determined whether an adversary is able to generate corresponding pairs of plaintext and ciphertext to use as a lever in compromising secret keys. The ideas are illustrated by analysis of two well-known authentication systems which have been used in practice. They are Kerberos and KryptoKnight. It is shown that for the Kerberos protocol, an adversary can acquire at will an unlimited number of known plaintext-ciphertext pairs. Similarly, an adversary in the KryptoKnight system can acquire an unlimited number of data pairs which, by a less direct means, can be seen to be cryptanalytically equivalent to known plaintext-ciphertext pairs. We propose new protocols, using key derivation techniques, which achieve the same end goals as these others without this undesirable feature.

1 Introduction

In recent years great advances have been made in understanding how to design cryptographic protocols for entity authentication and secure message exchange. Various techniques have been proposed (e.g., [19, 20, 12, 13, 10, 4]) and thereafter security flaws or weaknesses were discovered. To repeatedly find and fix problems in the published authentication mechanisms is an active research topic. Meanwhile, systems based on these mechanisms have been implemented; two well-known systems are Kerberos [17, 14] and KryptoKnight [18]. Naturally, these implementations should also be frequently examined and debugged in accordance with discoveries of the problems in the underlying mechanisms.

In applications of distributed computation which crucially require secure communication, entity authentication establishes a secure channel between communication parties remotely situated in a hostile environment. In the techniques

* This work is funded by the UK Engineering and Physical Sciences Research Council under research grant GR/G19787.

mentioned above, this task is achieved through the use of a trusted authentication server and as a special case, the server itself can be one of the communication parties. It is always assumed that a secure channel already exists between a client principal and the server. Let this existing channel be referred to as a *long-term channel*, which is established through some expensive method in a higher level of the security hierarchy. It must be understood that the security essence of a long-term channel is its low bandwidth: its usage must be limited only to establish other channels of higher bandwidth. In other words, authentication protocols are meant to use a secure long-term channel to transmit or to agree a *small* amount of secrets (usually, a cryptographic key) which may serve as a new secure channel (called a *session channel*) along which information can be transmitted with a smaller delay. Notice that a channel with a high bandwidth is vulnerable to temptations in terms of cryptanalysis; it therefore should a limited lifetime. Whenever needed, communication parties should run an authentication protocol to create a new session channel.

It is thus clear that the reason for maintaining the low bandwidth of a long-term channel is in order to foil cryptanalysis passively and/or actively targeted on it. Only by taking this into account does the required and assumed long lifetime of a long-term channel make sense. In public-key cryptographic techniques there is also a need for thoughtful use of a long-term channel. For instance, in the case of the RSA algorithm [21], a long-term channel between a pair of principals can be identified with the private keys of each party; such a key is matched to the public key which is *certified* to the principal. This viewpoint should be considered when the RSA algorithm is used to "bootstrap" a conventional encryption scheme.

In this paper, we keep in mind the working principle of entity authentication mechanisms discussed above while we investigate the existing techniques. We focus on two implemented systems, Kerberos (Section 2) and KryptoKnight (Section 3). These two systems will be shown to allow an adversary to acquire at will an unlimited number of known plaintext-ciphertext pairs. So viewed by the adversary, long-term channels of these systems are actually used at a *very high* bandwidth, even higher than that of any session channel. This is inconsistent with respect to the working principle of authentication mechanisms. Our investigation will result in some insight into how an authentication mechanism should be designed to fulfill the intended purpose of entity authentication. In Section 4, we will demonstrate our idea by presenting remedies for the problem, using an idea of "one-time" channel derivation, which achieves the same end goals as these others without the undesirable feature revealed. In addition we will see another good feature possessed in one of our remedy techniques: *perfect forward secrecy* [8], which means that loss of a long-term key should not lead to loss of any session key which has been established by the lost key. We will also discuss the possibility of extending our techniques to conventional authentication protocols. Finally, Section 5 forms our conclusion.

The remainder of this section is devoted to a brief overview of cryptanalysis threats that we will be discussing throughout the paper. Further details may be found in various texts such as the recent book by Schneier [22].

1.1 An Overview of Cryptanalysis Threats

In conventional cryptography the sender and recipient of a message share a key, which is known to no other principals, that allows each of them to encrypt or decrypt messages. It is inevitable that an attacker will be able to record and analyse a large amount of the encrypted ciphertext transmitted between the two parties with the aim of extracting the plaintext or analysing the key used. The cryptographic algorithm employed should be designed so that this is not possible for an attacker using the resources anticipated.

However, resistance to such a *ciphertext only* attack is not sufficient to guarantee security. It may be anticipated that the attacker will be able to obtain portions of ciphertext for which he also knows the corresponding plaintext. These are known as *plaintext-ciphertext pairs.* A *known plaintext* attack, which attempts to find the shared key from a number of plaintext-ciphertext pairs, is considerably harder to defeat. An even sterner case is the *chosen plaintext* attack, in which the attacker is able to choose plaintext portions and see their encrypted versions. Recent advances in cryptanalysis [3] have shown that resilience to known and chosen plaintext attacks is not so easy to achieve as had been previously thought. It is particularly worth noticing that authentication protocols which apply a *challenge-response* technique, if not carefully designed, can be abused to form a substantial amount of plaintext-ciphertext pairs. In Section 2 we will see that the authentication system Kerberos, which implements the techniques of a category of published authentication protocols, allows an attacker to obtain an unlimited amount of plaintext-ciphertext pairs to be used to undermine the long-term channel between a server and a client.

Another type of cryptographic function is a *one-way hash function.* Such a function has a one-way property which means that it is easy to compute hashed values in the direction from domain to range but computationally infeasible, given almost any hashed value, to find any input it could have come from. Typically such functions map long strings onto much shorter ones. An attractive feature due to the unequal sizes of domain and range is that plaintext-ciphertext pairs generated on the channel are of little use for an opponent. The idea of using one-way hash functions as the basis of cryptographic protocols appeared quite early in the literature, e.g., Evan *et al.* [9], Merkle [16] and Gong [10]. Subsequently the idea has been employed in the authentication and key exchange system KryptoKnight [18]. However, in Section 3 we will see that owing to an undesirable design feature, an attacker can force the domain and the range of the one-way function implemented in the system to have the same size and at the same time obtain an unlimited amount of plaintext-ciphertext pairs. Normal cryptanalysis techniques can then be applied to undermine the long-term channel.

Having explained the threat scenario, it is attractive if we can *guarantee* that a long-term channel will never be used to provide plaintext-ciphertext pairs. Such a technique will be presented in this paper.

2 Kerberos

Kerberos is based on the Needham-Schroeder protocol, but makes use of timestamps as nonces to remove the problem pointed out by Denning and Sacco [6]. In Kerberos, basic message exchanges between a network client A and an authentication server S have the following form:

1. $A \rightarrow S$: *request*
2. $S \rightarrow A$: *reply*

In this presentation, the line $X \rightarrow Y$: Z describes a message communication directed from principal X to principal Y; Z represents the transmitted message. Requests from clients are always sent in plaintext and replies from the server are organised messages called *tickets*. A ticket is a record that helps a client to authenticate a service. A slightly simplified form of ticket can be written as below (cf., Kerberos version 5 [14]):

ticket = version-number, addresses, names, *encrypted-part*

where the *encrypted-part* is as below:

encrypted-part = {flag-bits, session-key, address, names, timestamps, lifetimes, host-addresses, authorization-data $\}_{K_{AS}}$

The notation $\{M\}_K$ denotes a ciphertext generated from a (symmetric) crypto-algorithm which uses M as input data and K as encryption key. In the above example, the key K_{AS} is the secret key shared between the client principal A and the server principal S; it is the basis of the long-term channel existing between these two principals.

We see that in the encrypted part of a Kerberos ticket the messages are non-secret data except for the session key. To an external adversary, the principal addresses and names are fully known, and timestamps and lifetimes have easily guessable formats. To an internal adversary, such as the third party principal, B, with whom A intends to share a session key by initiating an authentication run, all data in the encrypted part of a ticket are fully known. (In this paper, B is always viewed as a potential enemy.) We observe that the encryption algorithms used by Kerberos (they will be discussed below) treat these data as secrets. We regard such a treatment to be unwise. In so doing, each run of the protocol will generate plaintext-ciphertext pairs for an adversary to analyse the long-term secret key shared between the server and a client principal. We now explain why in the case of Kerberos, the effect of the cryptanalysis may not be ignored.

In Kerberos, a request from a client principal to the server is sent in plaintext. Thus the adversary's action is not limited only to passive monitoring of the normal runs of the protocol on the network traffic, which can only allow him to obtain a trivially small amount of plaintext-ciphertext pairs. The opponent can in fact masquerade as A and send an *unlimited number* of plaintext requests

to S, who presumably is a node in the computer network and will prompt the opponent by supplying tickets, i.e., plaintext-ciphertext pairs, onto the network.

In the encrypted part of a ticket, known data follow a session key which varies in every ticket returned from the server. Thus in the case of using a chained encryption algorithm (in Kerberos V5, the encryption algorithm used is cipher block chaining (CBC), see Section 6.3 of [14]), the constant known plaintexts will be "garbled" by the feedback of the previous ciphertext output. We now look at how such a garbling will help the adversary to obtain a large amount of plaintext-ciphertext pairs. The output of a block cipher using CBC mode is a sequence of n-bit cipher blocks which are chained together in that each cipher block is dependent not only on the current input plaintext block, but also on the previous output cipher block. Let $P_1, P_2, \cdots, P_m$ be plaintext blocks to be input to CBC algorithm and $C_1, C_2, \cdots, C_m$ be ciphertext blocks output from the algorithm. Then the encryption procedure to generate a block of ciphertext is as below:

$$C_i = eK(P_i \oplus C_{i-1})$$

where $eK()$ denotes an encryption algorithm keyed by K and $\oplus$ denotes the addition, bitwise modulo 2. So $P_i \oplus C_{i-1}$ and C_i form a plaintext-ciphertext pair. Now let P_1 be the session key which varies in every ticket returned from the server. Then it is easy to see that C_1, C_2, $\cdots$ vary in every ticket. The consequence of this garbling is: simply repeating a constant request, the opponent will be guaranteed to obtain varied plaintext-ciphertext pairs with which he can build a dictionary. Notice that the correct use of CBC requires each encryption calculation be initialised by a new "initial vector" (IV); these varied IV's can also play the role of the session keys.

It seems there is a simple cure for the problem that we have revealed: the server should record the requests from clients; if numerous requests from a principal are detected within a short period of time, the service should be denied. However, this then allows a denial of service attack with which a malicious person can cheat the server to stop serving innocent clients. A denial of service attack of this kind can only be prevented if such an attack as the above is allowed.

From our analysis so far it is apparent that the Kerberos authentication system can be abused by an opponent to obtain an *arbitrary* amount of plaintext-ciphertext pairs. It is not hard to imagine that by performing the attack in a short period of time, the amount of *pairs* gathered by the opponent can exceed the quantity of ciphertexts of a session. This forms rather a strange situation: a session key which generates a *smaller* amount of *ciphertext-only* data is stipulated to have a short lifetime while a key which can generate a *larger* amount of *plaintext-ciphertext pairs* is, on the contrary, to be used in a much longer period of time. Considering that cryptographic keys in modern encryption algorithms (such as DES) have a fixed format, it cannot be that some keys are unconditionally stronger than others. The stipulated difference in lifetimes of the keys is due to the consideration of different types of data to be encrypted. Unfortunately, in the case of Kerberos, this reasonable stipulation turns out to be a dangerous practice.

3 KryptoKnight

KryptoKnight is an authentication system developed by Molva *et al.* [18]. Its technical basis is similar to that of a protocol that Gong devised [10]. In the treatment of non-secret data, KryptoKnight is extremely different from Kerberos where a substantial amount of non-secret data are encrypted against access. In KryptoKnight, non-secret data are sent in plaintext. The integrity of these data is protected by using the one-way property of a cryptographic transformation. Such a treatment shows a better understanding of authentication mechanisms, i.e., the required property of cryptographic services for authentication is one-way transformation, rather than secret concealment. In the previous section we have seen that to conceal non-secret data against access is not a good practice. Authentication applying the one-way property has many other advantages over applying the secret-concealment property [15].

In the message exchange for key distribution, KryptoKnight uses the image of a one-way transformation, namely, a mechanism called a message authentication code (MAC), as a key to conceal a distributed session key in the fashion of the one-time pad. Below is such an exchange where a client principal A requires the authentication server S to generate and send to her a session key K to be shared with a third party B.

1. $A \rightarrow S : A, B, N_A$
2. $S \rightarrow A : N_S, N_A, B, T, MAC_{K_{AS}}(N_A \oplus B, N_S, N_A \oplus S, T) \oplus K$

In the above messages, N_A is a nonce chosen by A for verifying the timeliness of the message replied from the server, N_S is the nonce generated by the server and T is a lifetime stating the expiration time of the distributed session key K. The one-way transformation is denoted by $MAC_{K_{AS}}(\cdot)$; it is keyed by the long-term key K_{AS} shared between A and S. Notice that because of the one-way property of the MAC mechanism, the long string of plaintext input and the short string of the MAC (64 bits, see [18]) in message line 2 do not form useful pairs for an adversary. Furthermore, the MAC is concealed by the session key so it is not available to an external adversary on the network. Therefore, the attacking scenario that applies to Kerberos does not apply to KryptoKnight.

However, this clever design does not stop an internal opponent who has a long-term channel with the server. Assume that B is such a person. The message line 1 sent in plaintext means that B can masquerade as A, at the same time playing his own role. By fixing (or carefully choosing) N_A, he can obtain an unlimited number of MAC's which we put in the following set:

$$\{MAC_{K_{AS}}(N_A \oplus B, N_S, N_A \oplus S, T) \mid N_S \text{ known to } B\}$$

Notice that among the data input to these MAC's only N_S is a variable, or a real input value; the rest of the data are constants. Therefore we can rewrite the above set as the following one:

$$\{MAC'_{K_{AS}}(N_S) \mid N_S \text{ known to } B\}$$

In fact, elements in this set can be viewed as outputs from a block encryption algorithm which transforms one block of nonces, i.e., N_S, into one block of ciphers with the same block size. A more appropriate name for such a transformation should be ECB, the electronic code book mode of operation on a block cipher algorithm. Algorithmically, we can see little difference between a MAC with one-block length of input string and an ECB with the same input string. Now that both N_S and $MAC'_{K_{AS}}(N_S)$ are 64-bit blocks (see [18]), they form a perfect plaintext-ciphertext pair. B can build a dictionary of such pairs for undermining the long-term channel between A and S.

Similar to the scenario of the denial of service attack toward Kerberos that we discussed in the previous section, it is not a good solution to stop serving B when numerous malicious requests are detected. A desirable solution should be some mechanism designed in the protocol which does not prevent malicious action but instead prevents achieving the intended goal of such a malicious action. For instance, it will be attractive if no plaintext-ciphertext pairs will be produced against the long-term channel through sending a large amount of malicious requests onto the network. Such a technique will be devised in the next section.

4 Two Remedies for KryptoKnight

Kerberos and KryptoKnight apply encryption techniques in two extremely different manners. Kerberos overuses the secret-concealment property of crypto-algorithms; it unnecessarily, even harmfully, protects the confidentiality of non-secret data. KryptoKnight, on the other extreme end, underuses that property; lack of a secret transmitted along the long-term channel (the session key distributed in KryptoKnight is not a secret to an internal attacker as the third party) makes the channel too exposed. Naturally, we should consider a balance between these two extreme situations. Our remedies are to design some secrets to be passed through the long-term channel. Such a secret is protected by, and protects, the long-term key. Two remedies for KryptoKnight using this idea are given below.

4.1 Remedy Scheme 1

In the first remedy, the needed secret is a nonce N'_S replied from the server. The original protocol will be revised into the following version:

1. $A \rightarrow S : A, B, N_A$
2. $S \rightarrow A : \{N'_S\}_{K_{AS}}, N_S, B, T, MAC_{K_{AS} \oplus N'_S}(N_A \oplus B, N_S, N_A \oplus S, T) \oplus K$

In this specification, the usage of the identifiers is the same as in the original KryptoKnight, except the extra nonce N'_S, which is generated by the server for each run. The server sends it to A under the protection of the long-term key K_{AS} with an appropriate encryption algorithm (e.g., DES ECB). Thus, N'_S is a secret

between A and S. By adding this secret, bitwise modulo 2, to the long-term key K_{AS}, a "one-time" channel is formed and used to create a "one-time" MAC for each run. Thus, if B performs the same attack that we have described in the previous section, then each attacking run will give him *one* plaintext-ciphertext pair against a "*one-time*" channel.

Finally, it is not difficult to see that in this revision, except that we have eliminated the attacking scenario revealed in Section 3, the security essence of the original KryptoKnight has not been changed and this can be analysed analogously to that of the original KryptoKnight [18].

4.2 Remedy Scheme 2

In the second remedy, the needed secret is derived from the Diffie-Hellman exponential key exchange technique [7]. Briefly, A and S each pick random exponents R_A and R_S. Assuming they agree on a common base α and modulus p, A computes $N_A = \alpha^{R_A} \bmod p$ and S computes $N_S = \alpha^{R_S} \bmod p$. The N_A and N_S are used in the same way as these two identifiers are in KryptoKnight.

Now A, knowing R_A and $\alpha^{R_S} \bmod p$, can compute

$$(\alpha^{R_S})^{R_A} \bmod p = \alpha^{R_S R_A} \bmod p$$

Similarly, S can compute

$$(\alpha^{R_A})^{R_S} \bmod p = \alpha^{R_A R_S} \bmod p$$

So A and S agree a secret

$$Y = \alpha^{R_S R_A} \bmod p = \alpha^{R_A R_S} \bmod p$$

The value Y will be used as the needed secret which is exclusively shared between A and the server.

A message exchange for key distribution to realise this idea is given below.

1. $A \rightarrow S : A, B, N_A$
2. $S \rightarrow A : N_S, N'_S, B, T, MAC_{K_{AS} \oplus Y}(N_A \oplus B, N_S, N_A \oplus S, T) \oplus K$

The protocol presentation is almost identical to KryptoKnight. In the message returned from S, a one-time channel is used to create the MAC. This channel is formed by adding, bitwise modulo 2, the secret Y to the long-term key K_{AS}. It is one-time because even if N_A is a replay of an old message, the server will always generate a new N_S, and so Y is fresh. Now let B perform the same attack that we have described in Section 3. As above, each attacking run will give him *one* plaintext-ciphertext pair against a *one-time* channel. So no plaintext-ciphertext pair will be generated by this protocol against the long-term channel based on K_{AS}. In order to form a dictionary against the long-term channel, B faces the well-known difficulty of computing a large amount of discrete logarithms.

The challenge-response mechanism of the original KryptoKnight based on the exchange of freshness identifiers N_A and N_S is maintained, because now

these two identifiers are essentially fresh and random as long as R_A and R_S are. N_S is now no longer related to N_A as it is in the KryptoKnight protocol. For message integrity and authentication purpose, the value N'_S is now a cipher of N_S under the session key K, i.e., $N'_S = \{N_S\}_K$. The security essence of the original KryptoKnight will not be changed due to this revision and can be analysed analogously to that of the original KryptoKnight [18].

A good property possessed by this version of the remedy for KryptoKnight can be referred to as *perfect forward secrecy*. An authenticated key exchange protocol provides perfect forward secrecy [8] if disclosure of long-term secret keying material does not compromise the secrecy of the exchanged keys from earlier runs. Here in the revised KryptoKnight, the secrecy of the session keys established in the history of a long-term key depends on the various one-time secrets agreed between A and S. The secrecy will not be damaged as long as these one-time secrets have been properly disposed of. This property is inherited from the use of the Diffie-Hellman technique for derivation of the one-time secrets.

4.3 Discussion

The first remedy scheme given in Section 4.1 effectively eliminates the potential attack revealed in Section 3 which allows an adversary to accumulate an arbitrary amount of plaintext-ciphertext pairs against a long-term key. The remedy strengthens the original KryptoKnight in terms of disallowing algorithmic cryptanalysis methods based on numerous chosen or known plaintext-ciphertext pairs. However, we should point out that if the cryptanalysis technique is simply key-space search by brute force, then that remedy does not strengthen the original protocol. This is because the computing time needed for searching a key for the first remedy protocol and that for the original KryptoKnight only differ a polynomial function of the key size (it is reasonable to view the computing time as a function of the key size). For instance, guessing a candidate key K_{AS}, we can obtain a candidate nonce N'_S in the remedy protocol by a step of decryption which takes a polynomial time; then we can further test whether $K_{AS} \oplus N'_S$ is a correct keying value to have been used for creating the MAC, and this test is the basic computation of any key-searching algorithm. Now that the time for key search is an exponential function of the key size, the polynomial difference due to the remedy will not count.

In the second remedy scheme, due to the use of the Diffie-Hellman exponential key derivation technique, the key searching problem now faces computing discrete logarithms. The difficulty of this problem will depend on the size of the prime p used. For a properly chosen large prime, the best known algorithm to date has a sub-exponential complexity [5]; no polynomial time algorithm is known. However, it should be noted that there is a trade-off between the extra security gained and the consequent increase in system complexity.

There are various techniques for distributing or agreeing session keys, but the entity authentication steps that are inevitably needed in these techniques are often very similar. In practice these are mainly achieved by using shared secret keys or passwords in a conventional fashion (see e.g., [8, 11] for non-conventional

key agreeing ideas with conventional authentication methods). Similar to the problem found in KryptoKnight, the authentication parts of these techniques are found to have weaknesses for allowing cryptanalysis of shared keys or passwords. Our techniques proposed in this paper show a practical idea for preventing potential algorithmic cryptanalysis threats based on gathering numerous chosen or known plaintext-ciphertext pairs. In addition, the discussion supplied here points out that cryptanalysis threats in terms of brute-force key search needs to be countered by extra system complexities. Some authors [1, 2] have considered methods against brute-force searching for passwords; the remedy scheme 2 can be viewed as a different approach to a similar goal.

4.4 Applicability of the Strengthening Technique to Conventional Authentication protocols

Finally, we point out that the techniques supplied in this paper can be applied to strengthening conventional authentication and key distribution protocols which employ authentication servers. Here we show an example based on using the remedy scheme 2.

The weakness that we have discussed on Kerberos in Section 2 generally applies to conventional protocols. For instance, in the case of the Otway-Rees protocol [20] below:

1. $A \rightarrow B : M, A, B, \{N_A, M, A, B\}_{K_{AS}}$
2. $B \rightarrow S : M, A, B, \{N_A, M, A, B\}_{K_{AS}}, \{N_B, M, A, B\}_{K_{BS}}$
3. $S \rightarrow B : M, \{N_A, K_{AB}\}_{K_{AS}}, \{N_B, K_{AB}\}_{K_{BS}}$
4. $B \rightarrow A : M, \{N_A, K_{AB}\}_{K_{AS}}$

an opponent can repeat sending messages specified in the first line on to the network (he can do so by varying M and using any garbage for the cipher chunk), and B will thereby prompt messages specified in the second line. This malicious action results in an accumulation of chosen plaintext-ciphertext pairs in the same way as the attack on Kerberos explained above.

To make an example of a wide application of our technique, we suggest strengthening the Otway-Rees protocol into the following version:

1. $A \rightarrow B : M, A, B, N_A$
2. $B \rightarrow S : M, A, B, N_A, N_B$
3. $S \rightarrow B : M, N_{SA}, \{B, N_A, K_{AB}\}_{K_{AS} \oplus Y_A}, N_{SB}, \{A, N_B, K_{AB}\}_{K_{BS} \oplus Y_B}$
4. $B \rightarrow A : M, N_{SA}, \{B, N_A, K_{AB}\}_{K_{AS} \oplus Y_A}$

where

$$Y_A = \alpha^{R_{SA} R_A} \bmod p = {N_{SA}}^{R_A} \bmod p = {N_A}^{R_{SA}} \bmod p$$

and

$$Y_B = \alpha^{R_{SB} R_B} \bmod p = {N_{SB}}^{R_B} \bmod p = {N_B}^{R_{SB}} \bmod p$$

and α, p are appropriate elements in the Diffie-Hellman key-agreement technique.

5 Conclusion

Protocols need to be designed to take account of their implementation as well as their abstract security properties. We have shown how well known protocols allow attackers to obtain unnecessary assistance in obtaining plaintext-ciphertext pairs for use in cryptanalysis. Finally we have illustrated that simple steps may be taken to prevent such attacks which have very little computational cost to the legitimate users.

Acknowledgements

We would like to thank Paul Van Oorschot for helpful comments and suggestions on a draft of this paper which lead to the use of the Diffie-Hellman exponential key derivation technique.

References

1. R.J. Anderson and R.M.A. Lomas. On fortifying key negotiation schemes with poorly chosen passwords. Computer Laboratory, University of Cambridge (obtained from personal contact), 1994.
2. S.M. Bellovin and M. Merritt. Encrypted key exchange: Password-based protocols secure against dictionary attacks. In *Proceedings of the 1992 IEEE Symposium on Research in Security and Privacy*, 1992.
3. E. Biham and A. Shamir. *Differential Cryptanalysis of the Data Encryption Standard.* Springer Verlag, 1993.
4. R. Bird, I. Gopal, A. Herzberg, P. Janson, S. Kutten, R. Molva, and M. Yung. Systematic design of two-party authentication protocols. In *Crypto '91, LNCS*, 1991.
5. E.F. Brickell and A.M. Odlyzko. *Cryptanalysis, A Survey of Recent Results*, pages 501–540. IEEE Press, 1992.
6. D.E. Denning and G.M. Sacco. Timestamps in key distribution protocols. *C.ACM*, 24(8):533–536, August 1981.
7. W. Diffie and M.E. Hellman. New directions in cryptography. *IEEE Trans. Info. Theory*, IT-22(6):644–654, 1976.
8. W. Diffie, P.C. Van Oorschot, and M. Wiener. Authentication and authenticated key exchanges. *Designs, Codes and Cryptography*, 2:107–125, 1992.
9. E. Evan, W. Kantrowitz, and E. Weiss. A user authentication scheme not requiring secrecy in the computer. *C.ACM*, 17:437–442, 1974.
10. L. Gong. Using one-way function for authentication. *Computer Communication Review*, 19(5):8–11, 1989.
11. L. Gong. Authentication, key distribution, and secure broadcast in computer networks using no encryption or decryption. Technical Report SRI-CSL-94-08, SRI International, 1994.
12. ISO/IEC. N 739, DIS 9798-2, information technology - security techniques - entity authentication mechanisms - part 2: Entity authentication using symmetric techniques, 1993-08-13.

13. ISO/IEC. CD 11770-2: Key management, part 2: Key management mechanisms using symmetric techniques, 1993-10-03.
14. J. Kohl and C. Neuman. The Kerberos network authentication service (v5). Internet Archive RFC 1510, September 1993.
15. W. Mao and C. Boyd. Development of authentication protocols: Some misconceptions and a new approach. In *Computer Security Foundations Workshop VII.* IEEE Computer Society Press, 1994.
16. R.C. Merkle. Secure communications over insecure channels. *C.ACM*, 21:294–299, 1978.
17. S.P. Miller, C. Neuman, J.I. Schiller, and J.H. Saltzer. Kerberos authentication and authorization system. Project Athena Technical Plan Section E.2.1, 1987.
18. R. Molva, G. Tsudik, E. van Herreweghen, and S. Zatti. Kryptoknight authentication and key distribution system. In *ESORICS '92, LNCS 648*, pages 155–174, 1992.
19. R.M. Needham and M.D. Schroeder. Using encryption for authentication in large networks of computers. *C.ACM*, 21(12):993–999, 1978.
20. D. Otway and O. Rees. Efficient and timely mutual authentication. *Operating Systems Review*, Vol 21(1):8–10, 1987.
21. R.L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *C.ACM*, 21:120–126, 1976.
22. B. Schneier. *Applied Cryptography.* John Wiley & Sons, 1994.