Reducing Risks from Poorly Chosen Keys*

T. Mark A. Lomas, Li Gong,
Jerome H. Saltzer†, Roger M. Needham

University of Cambridge Computer Laboratory
New Museums Site, Pembroke Street
Cambridge CB2 3QG, England

August 1989

Abstract

It is well-known that, left to themselves, people will choose passwords that can be rather readily guessed. If this is done, they are usually vulnerable to an attack based on copying the content of messages forming part of an authentication protocol and experimenting, e.g. with a dictionary, offline. The most usual counter to this threat is to require people to use passwords which are obscure, or even to insist on the system choosing their passwords for them. In this paper we show alternatively how to construct an authentication protocol in which offline experimentation is impracticable; any attack based on experiment must involve the real authentication server and is thus open to detection by the server noticing multiple attempts.

1 Introduction

A common risk in authentication systems is password guessing. Although this risk is usually associated with stored-password systems such as UNIX [1], it also can apply to network authentication systems that utilize the Needham and Schroeder [Needham] model of temporary key distribution from a server that knows a personal key for each user. This risk arises because in order to make such a system convenient enough for user acceptability, the personal keys of individual users may be derived from passwords, and the users may choose those passwords. Self-chosen passwords are notoriously easy to guess [Morris], especially if a large number of different guesses (for example, all the words in an online dictionary) can be made mechanically and each such guess verified for correctness without raising an alarm.

For example, the Kerberos authentication system of M.I.T. Project Athena [Steiner] exhibits this risk, because the Kerberos Key Distribution Server enciphers its initial response packet using a key derived from the user's password. An attacker could passively record an initial response packet from a Kerberos server, and attempt to decrypt that packet using keys derived from a series of guesses as to the password value. The attacker can immediately determine whether or not a guess is correct because for a correct guess the resulting decipherment of the packet will produce recognizable data, such as the name of a network service or the time of day. Because the attacker can work privately, the only cost of an incorrect guess is the time wasted doing a decipherment that produces unintelligible data.

The common approach to reducing this risk is to encourage, exhort, or force users to choose passwords that are hard to guess. Since such passwords can also be hard to remember, this approach sometimes encounters user resistance.

In this paper we explore a different approach, assuming that poorly chosen passwords are a fact of life, and look instead for ways to improve an authentication protocol so that it doesn't matter. One inspiration for this approach is the observation that automatic bank teller machines generally use 4-digit numerical passwords, yet when properly designed they do not seem to be particularly vulnerable to password guessing. The reason is that although one could guess an ATM password in something less than 9,999 tries, the system will generally confiscate the user's banking card after the third wrong guess, and thereby terminate experimentation [MasterCard]. The key to the non-attackability of the small ATM password is that guesses cannot be

*Published in the Proceedings of the 12th ACM Symposium on Operating System Principles, *ACM Operating Systems Review*, 23(5):14–18, December, 1989.

†At the time of writing this paper Professor Saltzer was visiting the University of Cambridge. His usual address is Massachusetts Institute of Technology, Room NE43-513, 545 Technology Square, Cambridge, MA 02139, USA.

[1] UNIX is a trademark of AT&T.

verified in isolation; each guess must be tried by invoking a part of the system that is in a position to log incorrect guesses and raise an alarm. Our protocol improvements are intended similarly to insure that the only way the correctness of a guess can be verified involves some part of the system that is in a position to notice wrong guesses, log them, and raise an alarm.

A second inspiration for our approach is the observation that if one enciphers a completely random string of bits, an attacker has no way of verifying that he or she has discovered the key. Thus one protection against guessing of passwords is to formulate an authentication protocol in which keys derived from passwords encipher only data that is unpredictable by the attacker.

This paper describes a concept that we name *verifiable plaintext*. Verifiable plaintext is a type of message of which *known plaintext* forms a subset. We are not aware of previous works that emphasize the distinction between these two concepts. We propose an authentication protocol that avoids encryption of verifiable plaintext with easily-guessable keys. Since our primary purpose is to introduce a new, somewhat orthogonal, protocol design consideration, the protocol described here has been chosen for ease of discussion, rather than for minimization of message traffic or optimal protection against more esoteric, e.g., slicing attacks[2]. The same techniques used here should be similarly applicable to protocols that give priority to those considerations.

2 A Typical Attack

The following hypothetical example shows the kind of attack that we believe it is possible to guard against.

An eavesdropper has been monitoring network traffic and recording copies of all messages that form part of an authentication. One day he hears the following statement: "This system forces me to change my password so often that I just use the name of the day on which it last forced me to change it." If it is possible for the eavesdropper to determine the actual password, by testing in turn each of the seven possible values, using only the messages that he has already recorded, then the protocol should be considered insecure. If, on the other hand, the protocol has the property that the eavesdropper must reveal to an authentication server his knowledge (or lack thereof) for each guess then the eavesdropper risks discovery, and we consider the protocol secure against guessing.

[2] A slicing attack is possible when rearrangement of a piece of ciphertext has known consequences on the plaintext which it represents. In this paper we have chosen not to analyse such attacks.

3 Notation and Terminology

The protocols introduced in this paper require the use of encryption to hide the contents of messages from all but the intended recipient. In common with other papers on authentication we use the notation $\{m\}^k$ to indicate a message m encrypted using an encryption key k.

The word "plaintext" refers to the unencrypted message m and the encrypted form of the message is the "ciphertext".

To emphasize the risk and the techniques used to reduce that risk, we use the term *well chosen* to describe an encryption key chosen at random from a large key space, and the term *poorly chosen* to describe an encryption key derived from a user-chosen password. The distinction is based entirely on the (presumably low) probability of an attacker's successful guessing of a randomly chosen key, compared with the (presumably higher) probability of successful guessing of a user-chosen password.

4 Known Plaintext

The concept of *known plaintext* has historically been one of major interest to both cryptographers and cryptanalysts [Kahn]. If a cryptanalyst is presented with a piece of ciphertext and can predict all or part of the plaintext before the message is decrypted then that message is said to contain known plaintext. There are two ways by which this knowledge may be exploited: it may be possible to determine the unpredicted part of the message; it might also be possible to discover the decryption key that corresponds to that message.

The following is an example of known plaintext and a way by which it may be exploited:

The recipient of a series of letters notes that they always begin with a return address. One day he receives an encrypted message not intended for him. It is a reasonable assumption that this message also begins with the same return address; in fact somebody who has seen none of the preceding messages might also make the same assumption. The address is known plaintext. The concept applies to any such predictable information, even if the exact position of the information in the message is not known.

Let us assume that the person who encrypted the message is not very careful in the choosing of encryption keys. The previous three keys were "Angela", "Beryl", and "Christine"; if the unintended recipient of the message attempts to decrypt it with a series of women's names, starting with the letter "D", then he or she may find a key for which the decrypted message begins with the expected return address. It is very likely that this result indicates discovery of the correct key.

5 Verifiable Plaintext

The concept of *verifiable plaintext* is very similar to, but somewhat more general than, that of known plaintext. If a message contains information that is recognisable when decrypted, whether or not it was predictable in advance, then we describe the plaintext as verifiable. Since this verifiability is a characteristic of the plaintext rather than the key we choose to refer to the messages as containing verifiable plaintext.

As an example of a message that does not contain known plaintext, but does contain verifiable plaintext, consider a simple authentication protocol in which Alice attempts to convince Bob that Alice knows a private key that they have previously agreed upon, by choosing a 64-bit random number and sending Bob an enciphered message containing that random number and its complement. Bob can verify Alice's claim by deciphering the message with the private key, and adding the two random numbers to see if they total zero. This message, though not containing known plaintext, contains verifiable plaintext, and an eavesdropper who is familiar with the protocol can attack it in a way similar to that by which he would have attacked a known plaintext message. By attempting to decrypt the message using a series of guessed keys he can note those keys for which the resulting plaintext sums to zero. If more than one key satisfies this requirement then he may have to monitor further transactions but he will soon have sufficient information to verify his guess at the key.

We use the word "recognisable" in a very general sense. If one encrypts and sends a piece of verifiable plaintext, and in some other packet an intruder intercepts something that might be the encryption key, the intruder can verify that it is the key by trying it on the ciphertext and looking at the result. Such a key is recognisable and is also, therefore, verifiable. It is for this reason that we should be wary of encrypting keys, even strong keys, under a weak key. Note that if this second packet is encrypted under another key then this second key is also verifiable.

It should be noted that known plaintext forms a subset of verifiable plaintext since information that is known in advance is obviously verifiable. Any message that has specific properties that the attacker might know in advance, such as being an ASCII string, or being an integer with a small range but stored in a large field, is also verifiable. The most worrying aspect of verifiable plaintext from the point of view of a protocol designer is that an eavesdropper can check the correctness of a guess as to the encryption key without any on-line transactions with some entity that would notice, and be suspicious of, wrong guesses. This aspect makes it difficult for the system to recognise when someone is performing such an attack.

It is possible to be more relaxed about the use of poorly chosen passwords if we can remove verifiable plaintext from messages that are enciphered with predictable keys.

If a public-key encryption system is used [Diffie] then another form of verifiable plaintext attack is possible that allows the attacker to determine the plaintext without discovering the decryption key. An eavesdropper monitors one or more transactions using a known public key. For each possible value of the plaintext he or she computes the corresponding ciphertext and compares it with the ciphertext that was logged previously. A match indicates that the plaintext has been found.

This form of attack may be defeated by introducing a random number, which we call a *confounder*, into the message. A confounder is distinct from a nonce, which is a random number to be acted upon by the recipient, in that it has no purpose other than to confound such an attack. The value of a confounder may be ignored by the recipient of the message in which it appears.

As will be shown in the following section it is possible to construct a series of messages, no one of which is vulnerable in itself, but which together contain verifiable plaintext.

6 A Two-Message Protocol

We first exhibit a two-message transaction upon which an authentication protocol might be based.

Several existing private-key protocols [Needham, Voydock] contain a message pair that can be cast in the form:

i. $A \rightarrow B$: $\{n\}^k$
ii. $B \rightarrow A$: $\{f(n)\}^k$

Alice generates a random number n and encrypts it using a pre-arranged private key k. Bob decrypts the message, computes some agreed function of the number and returns the encrypted form of the result to Alice. The function $f()$ insures that the two messages aren't identical.

Neither of these messages considered alone contains known plaintext since both n and $f(n)$ are random numbers. But assuming that $f()$ is not a secret, then the pair of messages together contain verifiable plaintext and are subject to a guessing attack very similar to that described above.

Consider, however, how this exchange is improved if two different keys are used.

i. $A \rightarrow B$: $\{n\}^{k1}$
ii. $B \rightarrow A$: $\{f(n)\}^{k2}$

If a single transaction takes place then even an exhaustive search cannot determine the keys. If an eavesdropper records multiple transactions that use the same key

pair, then the eavesdropper could verify a correct guess of both keys. Thus the effort of guessing has increased to require exploring the product of two key-spaces, and if either of the two keys is well chosen, guessing has been inhibited. This product property is a very powerful way of hiding poorly chosen keys.

7 A Mutual Authentication Protocol

In the following protocol we observe the design convention of requiring the authentication server to generate session keys. There are at least two reasons for relying on a server for this function: high quality randomness is not easy for deterministic machines to implement; some encryption algorithms have the property that certain classes of keys are weaker than others and should be avoided. Even if a client is able to generate good random numbers, that client should not be required to recognise weak keys. Ralph Merkle pointed out that if this convention were relaxed on the assumption that clients are capable of generating high quality session keys, then the protocol might be simplified. It is also conventional, rather than send a password, to send the value of a function that depends upon the password. This gives a slight increase in protection at little cost.

We now present a protocol that allows a server S to mediate between two clients A and B to allow mutual authentication. This protocol does not minimize the number of messages transmitted, but it is simple and symmetric, and easier to explain than a minimal protocol.

i. A → S: { A,B $, na1, na2, ca, \{ta\}^{Pa}\}^{Ks}$
ii. S → B: A,B
iii. B → S: { B,A $, nb1, nb2, cb, \{tb\}^{Pb}\}^{Ks}$

iv. S → A: $\{na1, k \oplus na2\}^{Pa}$
v. S → B: $\{nb1, k \oplus nb2\}^{Pb}$

vi. B → A: $\{ra\}^{k}$
vii. A → B: $\{f1(ra), rb\}^{k}$
viii. B → A: $\{f2(rb)\}^{k}$

Where $na1, na2, ca, ra, nb1, nb2, cb,$ and rb are random numbers generated by the originator of the first message in which they appear. The key Ks is the public key (more on this in a moment) of the server. Pa and Pb are the password-derived personal keys of the clients A and B respectively. The server generates a session key k used by A and B to communicate. The values ta and tb are pieces of recognisable but non-repeating information such as local time, recorded with a precision greater than the maximum allowed client-server clock skew.

Messages i and iii are similar enough that a single explanation can apply to both. Client Alice generates three random numbers $na1, na2$, and ca, produces a piece of timely information, which could have originated only from Alice, namely the current time encrypted under Alice's personal key, and announces that she is Alice and wishes to talk to Bob. For simplicity, we assume that Ks is the public key to a public-key encryption algorithm, used only for the purpose of sending initial requests to the key distribution server. This single use of the public-key technique is interesting because it is used only to communicate with a public service, so only a single such key is required for a system.

The use of public-key encryption does not eliminate the need to retain a secret because it is only possible to prove the identity of somebody who holds some secret information. In our example protocol the server's secret is a decryption key; in the case of each client the secret is a password.

The server deciphers message i (iii) using its private key, and verifies the claimed identity of Alice (Bob) by deciphering $\{ta\}^{Pa}$ ($\{tb\}^{Pb}$). If that decipherment does not produce the current time (within the allowable clock skew) the server logs a failure; if it does produce a current time, it responds with message iv (v).

Message iv (v) contains $na1$ ($nb1$) as proof that message i (iii) was correctly decrypted. Note that if the server cannot decrypt the first message then message iv is not sent so key Pa is safe from attack. (A server concerned about minimizing information leakage might send back a dummy message iv (v) as a response to an error; such a dummy would presumably have no value to a guessing attacker.)

Message iv (v) contains $na2$ ($nb2$) for two reasons. The simpler of the two is to protect Alice (Bob) against an attack by Bob (Alice) who already knows the value of k; insider attacks are covered in the following section. The more important reason is to prevent messages vi and vii or messages vii and viii to be exploited to verify k, which in turn would allow Pa or Pb to be verified.

The messages vi and vii contain a challenge ra and response $f(ra)$ and therefore allow key k, if guessed, to be verified. Similarly messages vii and viii when taken as a pair contain verifiable plaintext enciphered with key k. Fortunately key k originated at the key-distribution server and can therefore be assumed to be well chosen, so a verifiable plaintext attack is not feasible.

8 Insider Attacks

There is one remaining loose end. Although one would like to assume that the communicating parties trust each other, it does not seem appropriate to extend that trust to sharing passwords, or even to trusting that Bob would not try to guess Alice's password with the

aid of the residue of a successful transaction. Such a safeguard would also ensure that Bob cannot cause Alice's key to be compromised by compromising his own.

Confounders *ca* and *cb* serve the purpose of defending Alice from Bob and vice-versa. As mentioned before, a confounder is simply random baggage that has no purpose but to confound an attacker. In this case confounders are of value because someone attempting to guess Pa or Pb could attempt to construct messages i or iii without being able to decrypt them. Consider message i′, a replacement for message i, which omits the confounder *ca*, and message iv, which is unchanged:

i′. A → S: { A,B,$na1, na2, \{ta\}^{Pa}\}^{Ks}$
iv. S → A: $\{na1, k \oplus na2\}^{Pa}$

Bob, knowing the value of k, guesses the value of Pa, decrypts message iv using this guess, computes the corresponding ciphertext value for message i′, and compares it with the intercepted copy of message i′. If the two ciphertext versions are identical, Bob's guess of Pa has been verified. The otherwise unused random number *ca* confounds such an attack, under the assumption that all of the bits of enciphered message i depend on all of the plaintext bits of that message. Confounder *cb* similarly protects Bob's password from guessing by Alice.

Note that the protocol includes a time-stamp. If the attacker does not know the time exactly, presumably that lack of knowledge increases the number of experiments needed to verify a successful guess by only a small amount. However, if the time-stamp is carried to a precision far greater than the attacker could know, then the low order bits of the time-stamp can also act as confounders.

9 Summary and Suggestions For Further Work

This paper has suggested a class of risk, guessing of poorly-chosen passwords, for which an authentication protocol may provide protection. It has also offered a conceptual framework based on verifiable plaintext to determine whether or not a protocol is susceptible to password guessing. Finally, it has demonstrated some examples of techniques for avoiding verifiable plaintext in an authentication protocol.

We have not investigated whether or not protection against the risk of poorly chosen passwords inherently makes a protocol more expensive. ¿From a quick glance at the techniques, one might conjecture that the length of messages (and thus of the amount of material encrypted) does increase. We do know, however, that the number of messages need not increase.

We have also not attempted to establish conclusively that it is necessary to use a public-key system for the initial message from the client to the key distribution server, although we conjecture that it is required, because a private key system would seem to require that the client be able to keep a well-chosen secret, an ability that we assume the client lacks.

Finally, we suggest the possibility of developing an automated checker that analyses a protocol and reports whether or not it contains verifiable plaintext. Such a checker would appear to be considerably simpler to construct than, say, an automatic proof that a protocol is complete and correct.

References

[Diffie] Diffie, W. and Hellman, M.E., "New Directions in Cryptography", *IEEE Transactions on Information Theory*, Vol. IT-22, No.6, November, 1976, pp.644-654.

[Kahn] Kahn, D., *The Codebreakers*, MacMillan, New York, 1967.

[MasterCard] "PIN Manual: A Guide to the Use of Personal Identification Numbers in Interchange", MasterCard International, Inc., September, 1980, Reprinted in *Cryptography: A New Dimension in Computer Data Security* by Meyer, C.H. and Matyas, S.M., John Wiley and Sons., 1982, pp.429-444.

[Morris] Morris, R. and Thompson, K., "Password Security: A Case History", *Communications of the ACM*, Vol.22, No.11, November, 1979, pp.594-597.

[Needham] Needham, R.M. and Schroeder, M.D., "Using Encryption for Authentication in Large Networks of Computers", *Communications of the ACM*, Vol.21, No.12, December, 1978, pp.993-999.

[Steiner] Steiner, J.G., Neuman, C., and Schiller, J.I., "Kerberos: An Authentication Service for Open Network Systems", Proceedings of the USENIX Winter Conference, February, 1988, pp.191-202.

[Voydock] Voydock, V.L. and Kent, S.T., "Security Mechanisms in High-Level Network Protocols", *Computing Surveys*, Vol.15, No.2, June 1983, pp.135-171.