Protecting Poorly Chosen Secrets from Guessing Attacks*

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

June 8, 1993

Abstract. In a security system that allows people to choose their own passwords, those people tend to choose passwords that can be easily guessed. This weakness exists in practically all widely used systems. Instead of forcing users to choose well-chosen secrets, which are likely to be difficult to remember, we propose solutions that maintain both user convenience and a high level of security at the same time. The basic idea is to ensure that data available to the attacker is sufficiently unpredictable to prevent an off-line verification of whether a guess is successful or not. We examine common forms of guessing attacks, develop examples of cryptographic protocols that are immune to such attacks, and suggest a systematic way to examine protocols to detect vulnerabilities to such attacks.

1 Introduction

In a security system that allows people to choose their own passwords, those people tend to select passwords that can be easily guessed [Morris 79]. These poorly chosen passwords are vulnerable to attacks based upon copying information (for example, the result of applying a one-way hash function to a password or of encrypting a message using the password as the encryption key) and experimenting off-line.

Such secret guessing attacks are often associated with stored-password systems such as that in the UNIX operating system. Often overlooked is the possibility of applying such attacks to messages passed over networks; in particular, such attacks are likely to succeed if, for reasons of user acceptability, an encryption key is derived algorithmically from a user-chosen password. We use the term "poorly chosen" to describe an encryption key derived from a user-chosen password. We use the term "well-chosen" to describe an encryption key chosen at random from a large key space. The distinction is based on the (presumably low) probability of an attacker successfully guessing a randomly chosen key, compared with the (presumably higher) probability of successfully guessing a user-chosen password.

For example, the Kerberos authentication system [Steiner 88] is vulnerable to guessing attacks:

*This paper appeared in *IEEE Journal on Selected Areas in Communications*, Vol.11, No.5, June 1993, pp.648-656.
L. Gong is with SRI International, Computer Science Laboratory, 333 Ravenswood Avenue, Menlo Park, CA 94025.
T.M.A. Lomas and R.M. Needham are with the University of Cambridge, Computer Laboratory, Cambridge, CB2 3QG, England.
J.H. Saltzer is with Massachusetts Institute of Technology, Laboratory for Computer Science, Cambridge, MA 02139.

the key distribution server encrypts its initial response packet using a key derived from a user's password; an attacker can record such a packet and attempt to decrypt it using keys derived from a series of guesses as to the password. The attacker can readily determine whether or not a guess is correct because for a correct guess the resulting decryption of the packet will produce recognizable data, such as the time of day or the name of a network service.

Guessing attacks are most effective when a large number of guesses, for example, all of the words in a machine-readable dictionary, can be made automatically and each guess verified, to see whether it was correct, without raising an alarm. In the example of Kerberos, the experimentation with numerous passwords can be done off-line. Therefore, a successful guessing attack is computationally feasible where a correct guess is verifiable and a failed attempt is undetectable.

A common, but we believe unacceptable, counter to this risk is to encourage users to choose passwords that are obscure, thus difficult to guess ([NBS 85]). Some systems choose passwords for their users, but because these passwords are likely to be unmemorable, this method is inconvenient to the users. Likewise inconvenient to users are long passwords. For example, personal identification numbers (PINs) for automatic teller machines (ATMs) are generally only four to six digits because longer PINs would probably be written down.

In an earlier paper [Lomas 89], we started to explore an approach in which we accepted poorly chosen secrets as a fact of life and looked instead for ways to improve the security of the system so that such secrets do not make the system vulnerable to guessing attacks. We described what we called verifiable plaintext, which is a type of message of which known plaintext forms a subset. Other writers have not, to our knowledge, explored the distinction between these two kinds of plaintexts.

In this paper we develop these ideas further. We suggest a more generalized concept of verifiable text and describe techniques to frustrate guessing attacks. Our aim is to ensure that data available to the attacker is sufficiently unpredictable, which prevents an off-line verification of whether a guess is successful or not. In other words, important messages contain enough redundancy for the intended recipients to accept them but insufficient redundancy for the attacker attempting an off-line guessing attack. The only way that a guess could be verified involves interaction with some part of the system that is in a position to react (as, for example, ATMs do [MasterCard 82]) to an excessive number of tries, notice wrong guesses, log them, and raise an alarm.[1]

To illustrate applications of our techniques, we construct Needham-Schroeder style authentication protocols [Needham 78, Needham 87] in which any guessing attack must involve interaction with the authentication server and is thus open to detection by the server. We first give a protocol similar to that in the earlier paper [Lomas 89] (that protocol was originally chosen for ease of discussion rather than for actual use.); we then propose a number of variations of the protocol that take into account practical considerations such as the minimization of message traffic, the use of nonces instead of timestamps, the adaptation for simple (and possibly one-way) identification instead of key distribution, and direct authentication in the absence of a trusted third party to act as an authentication server.

Finally, we sketch a systematic way of examining a protocol to detect vulnerability to guessing attacks. Although our discussion uses only examples of user-chosen passwords, our analysis is equally applicable in other applications, such as the use of ATM cards or "smart-cards". Our new proposal is inexpensive and maintains both user convenience and a high level of security at the

[1]Some systems (e.g., [DEC 89]) may even lie to the client after an excessive number of guesses so as to prevent an attacker from gaining useful information.

same time. We recommend it as a viable alternative to many of the existing practices in using poorly chosen secrets or passwords.

2 Guessing Attacks

It is outside the scope of this paper to discuss safeguards against more esoteric attacks, such as slicing attacks where rearrangement or processing of ciphertext has predictable consequences on the plaintext that it represents ([Stubblebine 92]). We assume that the cryptosystems used are resistant to these attacks. In particular, we assume that every bit of a ciphertext depends upon all plaintext bits.

We will use the notation $\{m\}_k$ to indicate the result of encrypting a piece of text, a message m, using key k. We shall consider symmetric key or conventional cryptosystems such as DES [NBS 77, Smid 88] as well as asymmetric or public-key systems such as RSA [Diffie 76, Rivest 78]. The word "plaintext" refers to the message m that is encrypted to form the "ciphertext" $\{m\}_k$; similarly, $\{m\}_k^-$ denotes decryption. The notation "," denotes concatenation; for example, m, n represents the concatenation of m and n. Further notations will be introduced where necessary.

2.1 The UNIX Password System

The standard UNIX password system uses a password file, `/etc/passwd`, that contains, amongst other things, the result of applying a hashing function to each user's password. Certain UNIX vendors (for example, Sun) allow an alternative password scheme, supposedly to improve security; we shall discuss the SunOS scheme in the next section. For each user's password p, a "salt" value s is chosen at random, and a one-way hash function $g()$ is applied to the password and the salt. The values s and $g(p, s)$ are both stored in the password file. By varying the salt, which would be known to an attacker, UNIX allows a choice of 4096 different hashing functions; for the purpose of this discussion, we can consider the password to have been hashed using a function $h()$ that incorporates the effect of the salt, i.e., $h(p) = g(p, s)$.

A guessing attack follows the same procedure as a legitimate password validation. An attacker guesses a candidate password p' and applies the hashing function $h()$. If the value $h(p')$ matches the stored value $h(p)$ then it is assumed that $p' = p$. If $h(p')$ does not match $h(p)$ the attacker tries the next candidate value of p' from a list of guesses, perhaps a dictionary that contains all words in the sender's language, plus a list of proper names, rock group names, and cartoon characters [Feldmeier 89, Klein 90]. The Internet Worm guessed passwords in the same way [Seeley 89]. A possible solution is to restrict access to the file `/etc/passwd`.

2.2 SunOS Secure NFS

Because of the known risks of the UNIX password system, the SunOS-4.0 documentation recommends that the file `/etc/passwd` be removed when the C2 security features are installed [Sun 88]. A public-key system is used to improve security in the SunOS network file system. Unfortunately, the system design allows the problem of poorly chosen passwords to contaminate the public-key system: the use of a poorly chosen password compromises a pair of carefully chosen public and private keys.

More specifically, SunOS-4.0 introduces the file `/etc/publickey` that contains a user's network name A, the user's public key $+Ka$, and the corresponding private key $-Ka$. This private key is encrypted using DES with a key derived from the user's login password p. When A logs in, the `login` or `rlogin` command prompts for the password and decrypts $\{-Ka\}_p$ from A's entry to obtain $-Ka$; since $+Ka$ and $-Ka$ are mutual inverses, the value $-Ka$ can be checked to validate that p was correctly supplied. Later the key pair $(+Ka, -Ka)$ is used to establish temporary keys for remote procedure calls.

Since the file `/etc/publickey` is publicly readable, an attacker can guess p (say p') and compute $-Ka' = \{\{-Ka\}_p\}^-_{p'}$. Now he can choose an arbitrary message x and check that $\{\{x\}_{+Ka}\}^-_{-Ka'} = x$; if so, it is highly probable that $p' = p$. Moreover, $-Ka = -Ka'$. The attacker can conduct as many additional tests as needed by choosing new values for x to eliminate the chance of a false positive test. A possible solution is to restrict access to the file `/etc/publickey` as for the file `/etc/passwd` in the UNIX example.[2]

This example shows that the vulnerability to guessing attacks can be contagious: a poorly chosen password leads to the compromise of an otherwise well-chosen private key. The effect of aggregation may also be significant: two systems that are not vulnerable when considered in isolation may become vulnerable when used in a related manner. Suppose one user has accounts on two systems; often he will choose the same password p for both. Suppose that data items $\{r\}_p$ and $\{r'\}_p$ are separately stored in the two systems, where r and r' are both random numbers. If r and r' are independently generated, an attacker can gain little from the items since they contain no redundancy to aid him in validating a guess as to the password. However, if the systems are related such that there is a known relationship between r and r' (say $r' = r + 1$), the attacker can decrypt both items and check whether the above equation holds in order to verify his guess. Thus the possibility of guessing attacks must be examined in this wider context. We will show in the following sections that it is possible to construct a series of messages, no one of which is vulnerable in itself, but which together may be vulnerable to attack. Such a series of messages is not just of academic interest; commonly used protocols contain sequences very similar to our examples and thus are likely to be open to attack.

2.3 The Kerberos Authentication System

The server's initial response to a user request contains recognizable information such as a timestamp t and the name or address of a network service S; before transmission, the response is encrypted under a key derived from a user's password p [Miller 88, Steiner 88]. Let us assume that the packet has the form $\{t, S, ...\}_p$; the actual order of the message fields is unimportant.

An attacker can record the server's response m, guess the password (let us call this guess p'), and decrypt the message to obtain $m' = \{\{t, S, ...\}_p\}^-_{p'}$. The decrypted message will contain, amongst other things, the values t' and S', which may or may not correspond to the values t and S. If they do not correspond, then the attacker can deduce that the guess p' was incorrect; if they do correspond, then it is highly likely that $p' = p$. Note that the exact value t might not be known in advance, but the attacker may be able to recognize a plausible range of values; the case for S is similar, because the attacker might know that S is supposed to be an ASCII string with a certain structure such as `cs.cam.ac.uk` even though he might not know the actual letters in advance.

[2]Another problem in a networked system is how to verify the authenticity of a password file obtained by remote mounting. We do not address this issue here.

3 Known Plaintext and Verifiable Text

The concept of known plaintext has historically been of major interest to both cryptographers and cryptanalysts [Kahn 67]. If a cryptanalyst is presented with a piece of ciphertext and can predict all or part of the plaintext before the message is decrypted, the message is said to contain known plaintext. Any predictable information may constitute known plaintext whether or not its position in the message was known in advance; for example, if one sends an encrypted message about this paper to someone else, we could reasonably guess that the message contains words such as "key" or "password".

Suppose an attacker has obtained $\{m, n\}_k$ and knows the value m. One way to exploit this information is to take a guess k', compute $\{\{m, n\}_k\}_{k'}^{-}$, and see whether the result contains m. A match indicates that he has possibly guessed correctly; a mismatch indicates that he has definitely guessed incorrectly. If a public-key cryptosystem were to be used, it is unlikely that he can guess the private key. Nevertheless, he knows $\{m, n\}_k$, m, and the public key k, so he can guess a value n', compute $\{m, n'\}_k$, and check whether this matches $\{m, n\}_k$. In this case if there was a match, he has definitely guessed correctly since otherwise decryption is not one-to-one.

Verifiable text, on the other hand, could be a plaintext, a ciphertext, or the result of a chain of computation that may or may not involve encryption or decryption. Recall that to discover a poorly chosen secret an attacker takes a number of guesses and, together with his knowledge of the system, performs computations to see whether he can rediscover something that he already knows or can recognize. Often if he discovers a match, the guess was correct with high probability.

Let $P\{y \mid x\}$ denote the probability that y is true given that x is true. Let t be a threshold probability that we shall explain later. For a poorly chosen secret s, v should be considered "verifiable text" if (1) the attacker can recognize v when he sees it, (2) there exists a function $f()$ such that $v = f(s)$ and it is feasible for the attacker to compute $f(s')$ for a sufficient number of values of s', and (3) $P\{s = s' \mid f(s) = f(s')\} >= t$.

It is easy to see how to attack s when given a corresponding value v. It is also apparent that both known plaintext and verifiable plaintext [Lomas 89] form subsets of verifiable text. A guessing attack is very worrying if it can happen off-line. The threshold t is a measure of the system's tolerance in this regard. If $t = 1$, then the attacker can verify his guesses entirely off-line; if $t = 1/n$ where n is the total number of plausible passwords, then an off-line attack is doomed to failure. When $1/n < t < 1$, the attacker may significantly narrow down the range of possible values of the secret using off-line calculations. A system is totally secure against such attacks only if $t = 1/n$.

4 Protection Techniques

To introduce the basic techniques, we first examine a simple two-message handshake transaction, which is often found in existing protocols [Needham 78, Needham 87, Voydock 83]. This pair of messages may be considered two ways: as a message and its checksum or a single message with sufficient redundancy known to the attacker. In practice techniques such as checksums are often used to help guarantee message integrity; also the formats of messages are usually predetermined [ANSI 86, Linn 89]. These structural properties generally increase the vulnerability to guessing attacks [Gong 90c]. When considering the pair as a single message with sufficient redundancy, redundancy is difficult to assess objectively. A message that can be recognized by one could appear to contain unrecognizable random data to another.

We use the notation $A \rightarrow B : m$ to indicate that A sends a message m to B. The two-message handshake transaction may be cast in the following form:

1. $A \rightarrow B : \{n\}_k$
2. $B \rightarrow A : \{f(n)\}_k$

A generates a random number n and encrypts it with a predetermined secret key k that is shared between A and B. B decrypts the message, applies an agreed upon function $f()$ to produce $f(n)$, and encrypts the result before returning it to A. The function $f()$ ensures that the two messages are not identical. The cryptosystem is symmetric.

Neither of these messages when considered in isolation would appear to contain known or verifiable information since both n and $f(n)$ are random numbers. Provided $f()$ is not secret, the pair of messages when taken together are vulnerable to a guessing attack; the attacker guesses the key, decrypts both messages, and checks whether the two values correspond. Consider, however, how this exchange is improved if two different keys are used.

1. A $\rightarrow$ B: $\{n\}_{k1}$
2. B $\rightarrow$ A: $\{f(n)\}_{k2}$

If a single transaction takes place, then even an exhaustive search cannot determine the keys, because for any guessed value for $k1$ there is very likely to be a corresponding value of $k2$ for which the messages will appear correlated. If the attacker records multiple transactions that use the same key pair, he could verify a correct guess as to the keys, but both keys must be guessed–he could not attack one key in isolation. Thus, after taking birthday-attacks into account, the effort of guessing has increased to exploring the larger of the two key spaces, and if either of the keys is well chosen, guessing would be inhibited.

Of course it would be unreasonable to ask a user to remember two keys, at least one of which is well chosen, since the reason that users tend to use poorly chosen keys is that they have difficulty remembering even one well-chosen key. However, let us assume that one of the participants, say B, is a computer (or a user equipped with a tamper-proof smart-card or even a portable computer) and can be expected to remember both keys. If a public-key cryptosystem is used, then one of these keys, say $k1$, can be the public key for the computer; $k1$ can be assumed to be well chosen and, since it is a public key, the user can safely write it down without having to worry about maintaining its secrecy. Key $k2$, for which secrecy is important, would be a symmetric key, probably derived from the user's password.

A public-key system may be subject to an attack where an eavesdropper guesses the plaintext, some of which may already be known or predicted, and encrypts the guess to compare with an intercepted message. Suppose in our example the function $f()$ is an increment, that is, $f(n) = n + 1$. The attacker guesses $k2'$ and decrypts the second message to obtain n'. By encrypting n' using the public key $k1$, he obtains a value $\{n'\}_{k1}$ that may be compared with $\{n\}_{k1}$, which was already intercepted. If the computed and intercepted messages match, the eavesdropper knows $k2' = k2$ and that the guess was successful.

This form of attack may be defeated by introducing a sufficiently large random number, which we call a *confounder*, into verifiable messages that are to be encrypted under a public key. A confounder is distinct from a nonce, a random number to be acted upon by the recipient, in that a confounder has no purpose other than to confound such an attack. The value of a confounder may be ignored by the legitimate recipient of the message in which it appears. Our modified protocol becomes the following:

1. A $\rightarrow$ B: $\{c, n\}_{k1}$
2. B $\rightarrow$ A: $\{f(n)\}_{k2}$

A generates confounder c. Now a guessing attack is inhibited because the attacker needs to guess either $k2$ and the corresponding private key of $k1$ or $k2$ and c to reconstruct message 1, both of which are infeasible activities. Note that guessing c would be useless if $k1$ is unknown to the attacker. In a sense, a confounder also plays the role of a one-time pad [Kahn 67] but has the distinct advantage that its value need not be agreed upon in advance by the communicating parties.

An alternative is to be more selective in our choice of the function $f()$. If $f()$ is a well-designed one-way hash function then, even given the value $f(n)$, an attacker should not be able to determine n. For such a scheme to be effective, the challenge n must be encrypted under a well-chosen key, while the result of the hashing function, $f(n)$, may be encrypted under a poorly chosen key. The risk with this alternative is that n and $f(n)$ are often not strictly random numbers; they contain recognizable data (to the attacker) such as a timestamp. Now the attacker may still attack the poorly chosen $k2$ using message 2 alone. To defeat this attack, let the symbol $\oplus$ denote the bit-wise exclusive-or operation, we could replace $f(n)$ with $(c \oplus f(n))$ in message 2. The attacker, however, may still attack by guessing $k2$ and reconstructing message 1. To completely defeat this line of attack (which is based on knowledge of $f(n)$), we use two independently generated confounders as follows:

1. A $\rightarrow$ B: $\{c1, c2, n\}_{k1}$
2. B $\rightarrow$ A: $\{c2 \oplus f(n)\}_{k2}$

Here B, before safely discarding $c2$, uses it to mask any redundancy in $f(n)$. While A, who knows $c2$, could still compute $f(n)$, the attacker could not attack message 2 in isolation. Confounder $c1$ prevents the attacker from reconstructing message 1. The above discussion also applies to poorly chosen secrets that are not used as encryption keys.

5 Authentication Protocols

We show in this section that the protection techniques could be applied to develop practical protocols, such as authentication protocols, that are resistant to guessing attacks. For simplicity, we use the term authentication to refer to both mutual and one-way authentication with or without key distribution.

5.1 A Mutual Authentication Protocol

In the following protocol, we choose to observe the design requirement that the authentication server, rather than the clients, generate all session keys. There are at least two reasons to rely upon the server to perform this function: high quality random number generators (suitable for generating cryptographic keys) are particularly hard and sometimes impossible (when no random seeds are provided) to implement on deterministic machines; some encryption algorithms have the property that some possible keys are vulnerable to cryptanalysis and should be avoided. Even if a client is able to generate good random numbers, that client may not also be required to recognize these keys. If this requirement may be relaxed, then the protocol may be simplified. We also observe the common convention that, rather than send a password, we send the value of a function that depends upon the password; this convention increases protection at little cost.

We now present a protocol that allows a server S to mediate between the two clients A and B to allow mutual authentication. This protocol is not minimal in terms of the number of messages, but is made symmetric for easier explanation. Recall that the symbol $\oplus$ denotes the bit-wise exclusive-or operation.

Demonstration Protocol.

1. $A \rightarrow S : \{A, B, na1, na2, ca, \{ta\}_{Ka}\}_{Ks}$
2. $S \rightarrow B : A, B$
3. $B \rightarrow S : \{B, A, nb1, nb2, cb, \{tb\}_{Kb}\}_{Ks}$
4. $S \rightarrow A : \{na1, k \oplus na2\}_{Ka}$
5. $S \rightarrow B : \{nb1, k \oplus nb2\}_{Kb}$
6. $A \rightarrow B : \{ra\}_k$
7. $B \rightarrow A : \{f1(ra), rb\}_k$
8. $A \rightarrow B : \{f2(rb)\}_k$

The values $na1, na2, ca, ra, nb1, nb2, cb$, and rb are random numbers generated by the originator of the message in which they first appear. For example $nb1$, which first appears in the third message, is a random number chosen by B. The key Ks is the public key of the server. The keys Ka and Kb are the keys of the clients A and B respectively, shared with S; if the owner of a key (i.e., A or B) is a person rather than a computer, the key would be derived algorithmically from the user's password. The server generates a session key k that will be used by A and B to communicate with each other. We assume that the server is trustworthy and will not leak client keys or masquerade as clients. The values ta and tb are pieces of recognizable but non-repeating information, such as the local time, recorded with a precision greater than the maximum allowed client to server clock skew.

The first and third messages are very similar, so a single explanation can apply to both. Client A generates the three random numbers $na1, na2$, and ca; produces a piece of timely information that could have originated only from A (such as the time shown by A's clock encrypted under A's personal key, Ka); and announces that he is A and wishes to talk to B. Ks is the public key used only for encrypting initial requests to the key distribution server. Only a single public key, used to communicate with a public service, is necessary in the whole system.

The server deciphers message 1 (or message 3) using its private key and verifies the claimed identity of A (or B) by deciphering $\{ta\}_{Ka}$ (or $\{tb\}_{Kb}$). If the result does not match the current time, within the allowable client-server clock skew, then the server should log a failure. If the time is correct, the server responds with message 4 (or 5); the server might choose to respond to a failed authentication with a random message of the same length to confuse an attacker.

Message 4 (or 5) contains $na1$ (or $nb1$) as proof that message 1 (or 3) was correctly decrypted. Since the server validates message 1 (or 3) before responding, an attacker would not gain useful information from message 4 (or 5) by sending a spurious message 1 (or 3).

Message 4 (or 5) contains $na2$ (or $nb2$) for two reasons. Without $na2$, for example, the attacker can guess Ka, decrypt message 4 to obtain k, and then decrypt messages 6 and 7 for verification. The other reason is concerned with insider attacks.

Although we would like to assume that the communicating parties A and B trust each other, it does not seem appropriate to extend that trust to sharing passwords, or even to trusting that each would not try to guess the other's password with the aid of the residue of a successful transaction. A safeguard that works under these circumstances would also ensure that neither could cause the compromise of the other's password by compromising their own; thus, the protocol protects against participants who are either malicious or merely incompetent.

Without $na2$, B (who knows k after message 5) can guess Ka and decrypt message 4 to verify his guess. In addition, confounders ca and cb also serve to defend A from B and vice versa. Consider message $1'$, a replacement for message 1, which omits ca, and message 4, which is unchanged.

$1'$. $A \rightarrow S : \{A, B, na1, na2, \{ta\}_{Ka}\}_{Ks}$
4. $S \rightarrow A : \{na1, k \oplus na2\}_{Ka}$

B, knowing the value of k, guesses the value of Ka; decrypts message 4 to obtain $na1$ and $na2$; reconstructs message $1'$; and compares it with the recorded copy of message $1'$. A match would verify B's guess of Ka. Confounder cb similarly protects B's password Kb from being guessed by A.

Messages 6 and 7 (and similarly messages 7 and 8) contain a challenge ra and response $f(ra)$ encrypted under the session key k. These challenge and response pairs seem vulnerable to a guessing attack on k. Fortunately, key k originated at the key-distribution server and can therefore be assumed to be well chosen. Here ra and rb might be replaced by timestamps.

Note that the protocol uses timestamps. 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 timestamps are carried to a precision far greater than the attacker could know, then the low order bits of the timestamp can also act as confounders. There is no reason why clocks used for authentication should bear any relation to the real time; if we synchronize our clocks not to the actual time but to a randomly chosen time, or if the clocks progress at an unusual rate, attacks may be more difficult. This is not just an idle suggestion; there are good security reasons why a clock used for authentication should not be turned back, even if it has accidentally drifted forwards.[3]

The inclusion of a timestamp in message 1 (and 3) has another unusual impact [Gong 90a]. Suppose that the timestamp is replaced with information that does not reflect the timeliness of the message. Now since S cannot determine whether message 1 is fresh, except by recording all such messages (which we assume to be impractical), an attacker may play back a previously recorded message 1 and get a new response. The attacker should then have received two different versions of message 4: $\{na1, k \oplus na2\}_{Ka}$ and $\{na1, k' \oplus na2\}_{Ka}$. The two session keys, k and k', generated by the server are likely to be different since S should choose a new session key each time. However, because both

[3]However, such a clock poses additional security risks even if it is not turned back [Gong 92].

messages contain $na1$, the attacker can guess Ka and decrypt both to see if the same value $na1$ emerges. Therefore, random numbers must not be reused, and S should be able to detect replays of messages 1 and 3 so as not to respond to them again. In practice, the server would probably store messages until a time interval greater than the maximum client-server clock skew has elapsed. As an alternative to timestamps we might choose to use a challenge-response scheme as shown later in the "nonce" protocol.

5.2 Reducing the Number of Messages

By rearranging the routes of the messages, we can produce a protocol similar to that of Otway and Rees [Otway 87]. Our protocol retains the nice feature of theirs that two nested Remote Procedure Calls (RPCs) suffice for implementation.

Compact Protocol.

1. $A \rightarrow B : \{A, B, na1, na2, ca, \{ta\}_{Ka}\}_{Ks}, ra$
2. $B \rightarrow S : \{A, B, na1, na2, ca, \{ta\}_{Ka}\}_{Ks}, \{B, A, nb1, nb2, cb, \{tb\}_{Kb}\}_{Ks}$
3. $S \rightarrow B : \{na1, k \oplus na2\}_{Ka}, \{nb1, k \oplus nb2\}_{Kb}$
4. $B \rightarrow A : \{na1, k \oplus na2\}_{Ka}, \{f1(ra), rb\}_{k}$
5. $A \rightarrow B : \{f2(rb)\}_{k}$

Now the challenge ra is sent from A to B in the first message even though the reply is not expected until the fourth message. The security analysis of this protocol is essentially the same as for the demonstration protocol presented above, despite the minor difference that ra is now sent in plaintext, because knowing ra does not help in guessing Ka or Kb.

We also present a protocol in the style of Kerberos (albeit in a much simplified form) that might serve as a more secure replacement while providing the full functionality of the Kerberos system. User A identifies himself and obtains a secret key k (a "ticket") to be shared between A and the ticket-granting service B. Afterwards A can obtain tickets for other services from B. We assume that the server is competent to generate well-chosen keys so k is secure against guessing attacks. Since B is a server, key Kb, which it shares with the server S, can also be assumed to be well chosen. Only key Ka is likely to be poorly chosen and must be protected. Here ts is the time at the server.

Enhanced Kerberos Protocol.

1. $A \rightarrow S : \{A, B, na1, na2, ca, \{ta\}_{Ka}\}_{Ks}$
2. $S \rightarrow A : \{na1, k \oplus na2\}_{Ka}, \{A, k, ts\}_{Kb}$
3. $A \rightarrow B : \{A, k, ts\}_{Kb}$

Again the security analysis is similar to that of the demonstration protocol.

5.3 Using Nonces

In some systems, we may not wish to assume the availability of synchronized clocks. Instead we can use the well known challenge-response technique to establish the timeliness of messages. To our compact protocol we could add an extra round of exchange between A and S in which S issues a nonce ns. S can now detect replays of those messages in which nonce ns is used in place of ta and tb. Note that A forwards ns to B in message 3. The security analysis is unchanged.

Nonce Protocol.

1. $A \rightarrow S : A, B$
2. $S \rightarrow A : A, B, ns$
3. $A \rightarrow B : \{A, B, na1, na2, ca, \{ns\}_{Ka}\}_{Ks}, ns, ra$
4. $B \rightarrow S : \{A, B, na1, na2, ca, \{ns\}_{Ka}\}_{Ks}, \{B, A, nb1, nb2, cb, \{ns\}_{Kb}\}_{Ks}$
5. $S \rightarrow B : \{na1, k \oplus na2\}_{Ka}, \{nb1, k \oplus nb2\}_{Kb}$
6. $B \rightarrow A : \{na1, k \oplus na2\}_{Ka}, \{f1(ra), rb\}_k$
7. $A \rightarrow B : \{f2(rb)\}_k$

5.4 Identification

There are certain transactions that do not require a session key to be issued and may not even require mutual authentication. For example, a user of an ATM machine needs to be identified but the rest of the transaction may take place unencrypted. A protocol for such circumstances could be very simple, and its security analysis is as before.

Identification Protocol.

1. $A \rightarrow S : A$
2. $S \rightarrow A : ns$
3. $A \rightarrow S : \{A, ca, \{ns\}_{Ka}\}_{Ks}$

This protocol is significant in that an ATM card (or credit card, entry card, electronic security pass, phone card, etc.) would need to store only public information (i.e., Ks). A correct PIN would allow the card to be used, while a stolen card is of little value to an attacker. Consequently, the manufacturing of such cards would be simple and cheap, because they need not always be made tamper-proof. Moreover, administration cost could be reduced because one smart-card need not be associated with any particular user.

5.5 Using Secret Public Keys

It is plausible that sometimes some users may not be able to remember S's public key Ks, even though they could write down the key on a piece of paper (which could be accidentally thrown away) or store it in a hand-held calculator (which may have a bad battery). Any of the previously discussed protocols could be easily converted to handle this situation, though the following protocol is derived from the compact protocol. The derivation is simply to let S send two public keys to A and B in message 2.

Secret Public Key Protocol.

1. $A \rightarrow S : A, B$
2. $S \rightarrow A : A, B, ns, \{Ksa\}_{Ka}, \{Ksb\}_{Kb}$
3. $A \rightarrow B : \{A, B, na1, na2, ca, \{ns\}_{Ka}\}_{Ksa}, ns, ra, \{Ksb\}_{Kb}$
4. $B \rightarrow S : \{A, B, na1, na2, ca, \{ns\}_{Ka}\}_{Ksa}, \{B, A, nb1, nb2, cb, \{ns\}_{Kb}\}_{Ksb}$
5. $S \rightarrow B : \{na1, k \oplus na2\}_{Ka}, \{nb1, k \oplus nb2\}_{Kb}$
6. $B \rightarrow A : \{na1, k \oplus na2\}_{Ka}, \{f1(ra), rb\}_k$
7. $A \rightarrow B : \{f2(rb)\}_k$

This protocol differs from the compact protocol only in that S sends two public keys in message 2, one of which (Ksb) is passed on to B in message 3, and that A would later use Ksa in message 3 and B would use Ksb in message 4.

Clearly this protocol does not require the pre-distribution of S's public keys. However, the protocol adds more constraints on the choice of public-key systems. First, Ksa (Ksb) must be kept secret between A (B) and S at all times. A could forget Ksa after its use, but should never reveal it, because if the key Ksa were to be publicized, then message 2 would contain verifiable text. The protocol gets its name from this interesting secret way of using public keys.

For similar reasons, Ksa (Ksb) itself must not contain redundancy known to the attacker. The perfect public-key system should have the property that any random number of a suitable length can possibly be a public key. Moreover, encrypting random messages–messages that include confounders–must not leak information about the encryption key. Finding such a perfect system may be difficult. However, traditionally, public-key systems are evaluated against chosen-plaintext attacks, under the assumption that public keys are always known to the attacker. This assumption is no longer true in the above protocol, thus some public-key systems that were previously regarded as unfavorable may well be usable here, and fresh evaluation of public-key systems may be desirable.

If an imperfect system is used, the use of many different Ksa's might enable the attacker to narrow down the search space of Ka. Fortunately, S could conceivably remember one public key for each user, just like remembering the passwords, and always send the same public key to the same user. This defense also has the side benefit that S need not generate new public keys for each session. A replay of an old Ksa will merely lead A to send a useless message 3, but cannot cause security breaches. The remaining security analysis is identical to that of the compact protocol.

Finally, we note that such an alternative protocol could be used to replace the compact protocol, or both protocols could be made available in a system and the appropriate one be invoked as needed.

5.6 Direct Authentication

In some environments, a trusted party S may not be widely available. In this case, A and B may already share a poorly chosen secret (say Kab), and they wish to establish, in a secure way, a well-chosen session key. We could let A and S be one and derive from the secure public key protocol a direct authentication protocol. Here $Kab1$ is a public key and k is the session key, both chosen by A.

Direct Authentication Protocol.

1. $A \rightarrow B : ra, \{Kab1\}_{Kab}$
2. $B \rightarrow A : \{B, A, nb1, nb2, cb, \{ra\}_{Ka}\}_{Kab1}$
3. $A \rightarrow B : \{nb1, k \oplus nb2\}_{Kab}$
4. $B \rightarrow A : \{f1(ra), rb\}_k$
5. $A \rightarrow B : \{f2(rb)\}_k$

This protocol has the same constraints on the choice of a public-key system as that in the secure public key protocol. Since A is unlikely to be able to remember well-chosen secrets, A may need to generate a new public key $Kab1$ for every session.

Sometimes it may be desirable to let both A and B contribute to the choosing of the session key k. This could be arranged by letting each of them choose a key (say $k1$ and $k2$) in a way similar to the direct authentication protocol and use $h(k1, k2)$ as the session key, where $h()$ is a suitable one-way hash function.

In addition to the secret public key protocol, we could also convert other previously discussed protocols for direct authentication, but then we would assume that a party knows the other party's public key in advance.

6 Detecting Vulnerability

Given the widespread use of poorly chosen secrets, we would like to develop a systematic way by which protocols may be examined to see whether they are vulnerable to guessing attacks. Our approach is to gather all the relevant messages (from one or more executions of a protocol or a group of protocols) that might be available to an attacker and assume that his initial targets are certain potentially poorly chosen secrets, then search to see whether any of the messages contain information that might serve to verify a guess. We sketch this approach below.

First we specify the set M, which contains the messages, their components, and any keys used to construct the messages. For example, if we are considering the message $\{A, t\}_k$, then M should contain each of the components (A and t), the key k, and the message $\{A, t\}_k$ itself.

We then construct a set of rules R that show what is feasible to derive by combining elements of M. For example, from A, t, and k we may derive $\{A, t\}_k$; from A, t, and $\{A, t\}_k$ we may or may not be able to derive k depending upon assumptions of the cryptosystem. In other words, since each rule represents a computation that has a certain complexity, we could discard a rule that corresponds to theoretical deduction that is actually computationally infeasible to an attacker. Alternatively we could keep the rule and mark its complexity as infinity, just in case future developments make it feasible. A typical rule within R might look like $(A, t, k \rightarrow \{A, t\}_k)$.

Set R must be exhaustive and could be constructed in a systematic manner. Initially we start with an empty set to which we will add rules as we derive them. For a relevant computation $x = f(x_0, x_1, \ldots, x_{n-1})$, we can add the rule $(x_0, x_1, \ldots, x_{n-1} \rightarrow x)$ and, if $f()$ is invertible, we may be able to add rules $(x_0, x_1, \ldots, x_{i-1}, x, x_{i+1}, \ldots, x_{n-1} \rightarrow x_i)$ for various values of i. Conventional encryption $y = \{x\}_k$ may be characterized by the rule $(x, k \rightarrow y)$ and decryption $x = \{y\}_k^-$ by $(y, k \rightarrow x)$; the rule $(x, y \rightarrow k)$ would indicate that the cryptosystem is subject to known-plaintext attacks, which we expect not to be the case. Public-key cryptosystems can be characterized in a similar manner: if $y = \{x\}_{+k}$ then $(+k, x \rightarrow y)$ and $(-k, y \rightarrow x)$. Certain public-key systems [Rivest 78] have the property that $\{\{x\}_{k-}^-\}_{k+} = x$, leading to an additional rule $(+k, y \rightarrow x)$. The rule $(+k, -k \rightarrow 1)$ reflects the fact that $+k$ and $-k$ are mutual inverses, and so each may be used to verify the other.

The set W contains messages the attacker is expected to know; in particular, it should contain all messages that an eavesdropper might record. Initially we choose this set to contain those "messages" (elements of M) that the attacker is likely to know. For every rule $(x_0, x_1, \ldots, x_{n-1} \rightarrow x)$ within R, if W already contains all the x_i's then we should add x to W. In other words, the final state of W should be the closure of its initial state under R; there are well-known algorithms to perform this operation [Aho 74]. Last we specify the set G containing all secrets that an attacker might choose to guess. The sets W and G might differ between different assumed attackers.

Let V be the set of verifiable texts within M. By definition, any value x is verifiable if there is a rule $(x_0, x_1, \cdots, x_{n-1} \rightarrow x)$ in R such that every $x_i \in W \cup G \cup V$ and not every $x_i \in W$. For example, assume $A \in W$, $t \in W$, and $k \in G$, then $\{A, t\}_k$ is verifiable. Thus one candidate value of set G determines a unique candidate value of verifiable text x. Suppose two candidates of G, g and g', determine two corresponding candidates values of x, t and t'. Vulnerability is detected if an x is found such that $x \in W \cap V$ and $P\{g = g' \mid t = t'\}$ is sufficiently large. The attacking line in this case is to guess a secret, compute the corresponding value of x, and compare it with the recorded value (since $x \in W$). If they match, the probability that the guess is correct is $P\{g = g' \mid t = t'\}$.

Searching for verifiable texts can be cast as the path-finding problem in graph theory [Aho 74]. Let each message in M represent a node of a directed graph. Define all messages in $W \cup G$ to be reachable. Let rule $(x_0, \cdots, x_{n-1} \rightarrow x)$ define an unusual arc leading from a set of nodes $\{x_i \mid i = 0, \ldots, n-1\}$ to node x, and if every x_i is reachable, then x is also reachable. Now we would like to find out if there exists a path from nodes in $W \cup G$ reaching a node in W with a node in G on the path. The node at the end of the path corresponds to a verifiable text.

A path thus found corresponds to a possible way of attack. The algorithm can be expanded to find all such paths. Knowing these paths helps to improve the protocol design. Moreover, since each rule represents a computation that has a certain complexity, assigning this complexity as the length of the corresponding arc naturally defines the length of a path. Now $\mid G \mid$ times the length of the path is the complexity for the attacker to perform a guessing attack along that path. The shortest path gives a measure of the minimum effort for a successful attack.

Varying the initial set W helps to investigate the attacking power of a group of colluding participants. It also helps to reveal the effects of some messages being accidentally or deliberately leaked. Diverting vital messages to separate communication channels may prevent the attackers from recorded all messages necessary for the attack. In a sense, diversion resembles the concept of separation of privilege [Saltzer 75]. A more detailed discussion and some examples of analysis using this algorithm appeared elsewhere [Gong 90b]. In particular, we were able to detect insider attacking paths of a protocol that was resistant to outsider attacks.

7 Related Work

We are aware of only one protocol of a similar nature, recently proposed by Bellovin and Merritt [Bellovin 92], which achieves roughly the same objective as our direct authentication protocol. They also discussed the use of several cryptosystems in detail. Their protocol is as follows:

1. $A \rightarrow B : A, \{k\}_p$
2. $B \rightarrow A : \{\{r\}_k\}_p$
3. $A \rightarrow B : \{ra\}_r$
4. $B \rightarrow A : \{ra, rb\}_r$
5. $A \rightarrow B : \{rb\}_r$

A generates a public key k at random and encrypts it, using a symmetric-key cryptosystem, with their shared and poorly chosen password p as the key. B generates a session key r at random and encrypts this first under the public key k, received in the first message, then under the shared password p. The three remaining messages are a two-way handshake: A generates the nonce ra and B generates nonce rb; each expects to receive back their nonce from the other.

Despite the similarity in functionality, there is a clear advantage of our direct authentication protocol over theirs. Because of our use of confounders and nonces, the session key and the initial (poorly chosen) key are securely segregated such that the impact of compromising a session key (e.g., due to a large amount of traffic and successful cryptanalysis) is strictly limited to messages of that session. In their protocol, however, compromising session key r would enable the attacker to replay message 2 and masquerade as B in all future sessions. Moreover, it will permit a guessing attack; one can guess p, decrypt message 1 to obtain k, and use (p, k, r) to reconstruct message 2 for verification.

8 Summary and Suggestions for Further Work

We have shown that it is possible to arrange authentication and similar dialogues that are impracticable to attack by password guessing, even if it is rightly suspected that passwords are chosen from rather a small space. We would of course recommend that good passwords be used wherever possible. A systematic technique for checking vulnerability to guessing attacks is outlined; this systematic technique is valuable because vulnerability is not at all obvious to casual inspection, as we have found several times after devising example protocols.

It is not clear how much, if any, intrinsic cost is associated with making a protocol robust against guessing. Nor is it yet clear how important the use of public-key cryptography is; it would be pleasing if users did not have to retain huge constants, even if they are public. The public-key technique is however very straightforward.

References

[Aho 74] A.V. Aho, J.E. Hopcroft, and J.D. Ullman, *The Design and Analysis of*

Computer Algorithms, Addison-Wesley Publishing Company, Reading, Massachusetts, 1974, pp.195-201 and pp.299-300.

[ANSI 86] American National Standard Institute, "Financial Institution Message Authentication (Wholesale)", ANSI X9.9, August, 1986.

[Bellovin 92] S. Bellovin and M. Merritt, "Encrypted Key Exchange: Password-Based Protocols Secure Against Dictionary Attacks", in Proceedings of the IEEE Symposium on Security and Privacy, Oakland, California, May, 1992, pp.72-84.

[DEC 89] Digital Equipment Corporation, "VAX/VMS System Management: Guide to VMS System Security", Version 5.2, June, 1989.

[Diffie 76] W. Diffie and M.E. Hellman, "New Directions in Cryptography", *IEEE Transactions on Information Theory*, Vol. IT-22, No.6, November, 1976, pp.644-654.

[Feldmeier 89] D.C. Feldmeier and P.R. Karn, "UNIX Password Security - Ten Years Later", Proceedings of Crypto'89, published as *Lecture Notes in Computer Science*, No.435, Springer-Verlag, pp.44-63.

[Gong 90a] L. Gong, "Cryptographic Protocols for Distributed Systems", Ph.D. dissertation, University of Cambridge, April, 1990.

[Gong 90b] L. Gong, "Verifiable-Text Attacks in Cryptographic Protocols", in Proceedings of the IEEE INFOCOM '90, San Francisco, California, June, 1990, pp.686-693.

[Gong 90c] L. Gong, "A Note on Redundancy in Encrypted Messages", *ACM Computer Communication Review*, Vol.20, No.5, October, 1990, pp.18-22.

[Gong 92] L. Gong, "A Security Risk of Depending on Synchronized Clocks", *ACM Operating Systems Review*, Vol.26, No.1, January, 1992, pp.49-53.

[Kahn 67] D. Kahn, *The Codebreakers*, MacMillan, New York, 1967.

[Klein 90] D.V. Klein, "Foiling the Cracker: A Survey of, and Improvements to, Password Security", in Proceedings of the 2nd USENIX Unix Security Workshop, August, 1990, pp.5-14.

[Linn 89] J. Linn, "Privacy Enhancement for Internet Electronic Mail: Part I – Message Encipherment and Authentication Procedures", IAB Privacy Task Force, Network Working Group, RFC 1113, August, 1989.

[Lomas 89] T.M.A. Lomas, L. Gong, J.H. Saltzer, and R.M. Needham, "Reducing Risks from Poorly Chosen Keys", in Proceedings of the 12th ACM Symposium on Operating System Principles, Litchfield Park, Arizona, December, 1989. Published as *ACM Operating Systems Review*, Vol.23, No.5, pp.14-18.

[MasterCard 82] "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.

[Miller 88] 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, Massachusetts Institute of Technology, October, 1988.

[Morris 79] R. Morris and K. Thompson, "Password Security: A Case History", *Communications of the ACM*, Vol.22, No.11, November, 1979, pp.594-597.

[NBS 77] U.S. National Bureau of Standards, "Data Encryption Standard", U.S. Federal Information Processing Standards Publication, FIPS PUB 46, January, 1977.

[NBS 85] U.S. National Bureau of Standards, "Password Usage", U.S. Federal Information Processing Standards Publication, FIPS PUB 112, May, 1985.

[Needham 78] R.M. Needham and M.D. Schroeder, "Using Encryption for Authentication in Large Networks of Computers", *Communications of the ACM*, Vol.21, No.12, December, 1978, pp.993-999.

[Needham 87] R.M. Needham and M.D. Schroeder, "Authentication Revisited", *ACM Operating Systems Review*, Vol.21, No.1, January, 1987, p.7.

[Otway 87] D. Otway and O. Rees, "Efficient and Timely Mutual Authentication", *ACM Operating Systems Review*, Vol.21, No.1, January, 1987, pp.8-10.

[Rivest 78] R.L. Rivest, A. Shamir, and L. Adleman, "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems", *Communications of the ACM*, Vol.21, No.2, February, 1978, pp.120-126.

[Saltzer 75] J.H. Saltzer and M.D. Schroeder, "The Protection of Information in Computer Systems", *Proceedings of the IEEE*, Vol.63, No.9, September, 1975, pp.1278-1308.

[Seeley 89] D. Seeley, "Password Cracking: A Game of Wits", *Communications of the ACM*, Vol.32, No.6, June, 1989, p.700-703.

[Smid 88] M.E. Smid and D.K. Branstad, "The Data Encryption Standard: Past and Future", *Proceedings of the IEEE*, Vol.76, No.5, May, 1988, pp.550-559.

[Steiner 88] J.G. Steiner, C. Neuman, and J.I. Schiller, "Kerberos: An Authentication Service for Open Network Systems", in Proceedings of the USENIX Winter Conference, February, 1988, pp.191-202.

[Stubblebine 92] S. Stubblebine and V.D. Gligor, "On Message Integrity in Cryptographic Protocols", in Proceedings of the 1992 IEEE Symposium on Security and Privacy, Oakland, California, May, 1992, pp.85-104.

[Sun 88] Sun Microsystems, Inc., "Secure Networking", and, "Installing C2 Security Features", in *Security Feature Guide*, Part Number: 800-1735-10, Revision A, May 9th, 1988, pp.71-79 and pp.85-87.

[Voydock 83] V.L. Voydock and S.T. Kent, "Security Mechanisms in High-Level Network Protocols", *Computing Surveys*, Vol.15, No.2, June 1983, pp.135-171.

Li Gong was born in Beijing, P.R. China, in 1963. He received a B.S. (with honors, 1985) and a M.S. (1987) in computer science from Tsinghua University, Beijing, and a Ph.D. (1990) in computer science from the University of Cambridge, England. Before joining SRI, he was with ORA, Ithaca, New York, from May 1990 till February 1993. He was also a Visiting Scientist at Cornell University during academic year 1990-1991. His current research interest is in the theory and implementation of security and fault-tolerance in operating systems, distributed systems, and communication networks.

Mark Lomas is a Research Associate at The University of Cambridge Computer Laboratory where he has been studying for a PhD. His research interests include cryptography and security in distributed systems. He received his BSc (Hons) in Computer Science from The Hatfield Polytechnic (now The University of Hertfordshire). After graduating he worked at The ITT Europe Engineering Support Centre in Harlow.

Roger M. Needham graduated from Cambridge University in mathematics and philosophy in 1956, and then took the Diploma in Numerical Analysis and Automatic Computing in 1957. He has been in computing at Cambridge ever since, taking his Ph.D. in 1961. He has worked in operating systems, protection architectures, distributed systems, communications, and security. He succeeded Maurice Wilkes as Head of the Computer Laboratory in 1980, was promoted Professor in 1981, and elected to the Royal Society in 1985.

Jerome H. Saltzer was born in Nampa, Idaho, on October 9, 1939. He received the degrees of S.B. in 1961, S.M. in 1963, and Sc.D. in 1966, from the Massachusetts Institute of Technology, all in Electrical Engineering.

Since 1966, he has been a faculty member of the Department of Electrical Engineering and Computer Science, M.I.T., where he helped formulate the undergraduate curriculum in Computer Science, and developed the core subject on the engineering of computer systems. At the M.I.T. Laboratory for Computer Science he participated in the refinement of the Compatible Time-Sharing System (CTSS) and then was involved in all aspects of the design and implementation of the Multiplexed Information and Computing Service (Multics). More recently, his research activities have involved the design of a token-passing ring local area network, networking of personal computers, and designing the electronic library of the future. From 1984 through 1988 he was Technical Director of M.I.T. Project Athena, a system of networked engineering workstations for undergraduate education.

Professor Saltzer is a Fellow of the IEEE and the AAAS, and a member of: the Computer Science and Telecommunications Board of the National Research Council; the Association for Computing Machinery; Sigma Xi; Eta Kappa Nu; and Tau Beta Pi.