Authentication: a primer

Burton Rosenberg

December 30, 2002

1 Password Authentication

Authentication is the convincing of another that you are who you say you are. The most usual procedure for authentication is the presentation of a password. The password syllogism is:

```
Only Socretes knows the password
This man knows the password
Therefore this man is Socretes
```

In practice proof of knowledge of the password is provided by revealing it, generally typing it into the password box of the log-on screen. So not only Socretes knows the password, so does the authenticator. Or if the authenticator didn't know the password prior to authentication, it does so afterwards.

In the electronic version of this interchange, two other concerns arise. Since the authentication belongs to the near end of a communication channel, we must assure that the other end of the channel is not diverted after authentication; and we must assure that the channel is confidential, so no third party can eavesdrop and learn the password. Finally, the assurance of authentication is only a probability. Plato can, with a certain probability, guess Socretes' password, and then be accepted as Socretes.

$$\text{Socretes} \overset{\text{password}}{\longrightarrow} \text{Aristotle}$$

Figure 1: Password authentication

Whatever its deficiencies, password presentation remains a popular authentication mechanism. A variant of the password is th PIN's, a Personal Identifying Number, which is a short password drawn from the restricted alphabet of numeric characters. In applying either of these mechanisms, one should refer to the Federal Information Processing Standard FIPS 112 which makes recommendations and thereby establishes minimal standards for the use of PINs and passwords.

I am no lawyer, but FIPS 112 can be claimed as a standard of due care in the use of passwords. Security breaks which result in damage might be considered the software writer's negligence if security measures were not equal to accepted industry practice. It might be claimed that FIPS 112 provides a baseline for due care and accepted industry practice.

1.1 Password entropy

The authentication by password is only a probability. It is always possible for Plato to guess Socretes' password. The likelihood of this event should be quantified. If Socretes has n passwords to choose from, and he chooses one without bias, Plato's has a 1 in n chance of authenticating as Socretes purely by chance. The two key elements are the space of possibilities from which Socretes can choose, and his bias in making the choice.

Let $P = \{p_1, p_2, \ldots, p_n\}$ be the space of passwords, and by abuse of notation, p_i also be the probability that password p_i is chosen. The entropy of the space P is,

$$\mathcal{E}(P) = -\sum_{p_i \neq 0} p_i \log p_i.$$

Following Shannon, we take the base 2 logarithm so that the result is in units of "bits". If one of $256 = 2^8$ passwords is chosen uniformly at random, $\mathcal{E}(P) = 8$ bits, was we would hope.

Passwords are typically case sensitive and allow for punctuation characters, to increase the size of the password space, and should not be chosen as a common word, to reduce password bias.

Automated tools exist for password guessing. Authentication systems thwart on-line application of such tools by limiting in some manner the number of logon retries and reporting logon failures to a security log. These automated tools are used for off-line attacks, when a password file has been collected by some means and is in the possession of the attacker. Crack is the classic Unix program for this. L0pht-crack provides the NT system administrator with the same amusement for his machines.

Question: What is the entropy of user selected passwords?

1.2 The password hash

The subject requiring the authentication, the authenticator, must know the password. It is not usually stored on the authenticator without some encryption. If the database of encrypted passwords is stolen, the passwords are still safe until an attack is made against the encryption. Furthermore, the authentication decision is made by comparing encrypted versions of the password, so that passwords entered in the database are never again put clear, and perhaps it is infeasible to do so.

The traditional Unix password scheme passes the password through a modified DES encryption. Newer versions of Unix have replaced DES by other functions. FreeBSD uses an MD5 has of the password. Adoption of variants was delayed due to export considerations: strong encryption can be exported from the United States only under special license. FreeBSD solved this problem by distributing the code source with or without MD5 encryption. At this point in time, export controls have been eased and FreeBSD includes MD5 encryption in its base release.

The Unix DES algorithm is,

$$\begin{aligned} d_0 &= \mathrm{DES}^*_{h_0,p_1}(0) \\ d_i &= \mathrm{DES}^*_{h_0,p_1}(d_{i-1}),\ i > 0 \\ h &\overset{?}{=} h_0 | d_{24} \end{aligned}$$

The first two characters of the hash h, h_0, is called the salt. It is used to modify the DES algorithm, specifically, its E-box, becoming an additional part of the DES key, in a sense. The value zero is passed through the modified DES algorithm for 25 iterations, using the salt and the password as the encryption key. The result has the salt prepended and a check is made if the password hash is recovered. If so, authentication succeeds. If not, authentication fails.

The advantages of the salt are the following,

1. It slows down brute force password search by a factor equal to the size of the salt space.
2. It prevents easy identification of shared passwords by users on the same or different machines.
3. It prevents the use of standard DES hardware from participating in a brute force search for passwords.

Reference to the classic paper on Unix security.

Question: look at the code and get the exact bit description of the algorithm. In particular, what is the mapping from bits the characters?

Question: look at the FreeBSD code and describe its hash algorithm.

Windows NT also uses a password hash. In this case it is a simple MD4 of the password, yielding a 128-bit hash. However, NT uses this password hash in certain ways that makes it a complete replacement for the password. Under certain conditions, having the NT hash is as good as having the password, so their scheme does not truly increase security.

1.3 Trusted path, SAS, Trojans and other animals

There is a software and hardware path between the presentation of the password and the returned authentication decision. This path is trusted to be truthful by all components using the authentication decision. This trust is reasonable when the authentication system has a vested interest in providing accurate decisions for use by the other components. For example. the Unix authentication system shares with the user the password, but it would not share this password outside its system since the authentication system is part of the operating system that it is obligated to protect.

It is not easy to establish and maintain a trusted path. Network logon, for instance, requires the use of a network system which is outside the control of the authenticator. Even the integrity of this path for a local logon can be compromised. A Trojan is a program running on the machine which tricks the user into providing it with that user's password. It generally provides a false logon screen which the user believes to be genuine and collects the user's password.

An ultimate Trojan was an ATM placed in a shopping mall only for the purpose of collecting account numbers and corresponding PIN's. Users would commence a transaction, inserting their card and entering their PIN and the machine would then reject the request with some benign message such as "Machine out of order." The machine was collected after a day or two and the information gathered was extracted. *Reference.*

In practice, the authenticator authenticates the user but the user relies on extra-technological clues to authenticate the authenticator. The user also depends upon the true service to support them in the case their judgment fails. Banks have worked hard to maintain user's trust in ATM's and credit cards in order to preserve these valuable businesses.

Windows NT has a Secure Action Sequence, SAS, also known as Control-Alt-Delete, to establish a trusted path between the local keyboard and the Gina, NT's front-end to the authentication subsystem. The NT operating system makes strong guarantees that the host will respond to an SAS by presenting an authentic logon screen. Without such a guarantee, passwords can be gathered either by Trojans or by software keyboard sniffers. A software keyboard sniffer was used by the FBI in the Scarfo case *Reference.* Even with the SAS the path can be compromised, specifically between the wire connecting the keyboard to the host. Hardware keyboard sniffers are commercially available which will record all keystrokes as they pass from the keyboard into the host.

2 Challenge Response

Rather than present the password as proof of knowledge, the client can be asked to perform an action which implies knowledge of the password but which does not reveal the password. A Challenge-Response protocol asks the client to perform a calculation, a function of a randomly chosen number, the challenge, and a numeric version of the password, and to return the result of the calculation, the response.

The calculation should be such that the pair challenge—response assures knowledge of the password but does not reveal the password. The challenge must always be a new random number, else an attacker monitoring past sessions can present an old response to the old challenge and successfully authenticate.

$$\begin{array}{lcl} \text{Socretes} & \xleftarrow{\text{challenge}} & \text{Aristotle} \\ \text{Socretes} & \xrightarrow{\text{response}} & \text{Aristotle} \end{array}$$

Figure 2: Challenge response authentication

2.1 APOP

The Post Office Protocol (POP) allows network access to a user's email. Authentication is typically done using password authentication. The problem of password sniffing is particularly severe for POP for two reasons. First, POP is used for remote access while on travel, when the client is certainly unsure of the security of the network. Second, POP authenticates frequently, so the sniffer needn't be lucky to be looking at just the right moment.

APOP uses challenge response to alleviate these problems.

Question: a description of the protocol.

2.2 MS-CHAPv1

The documentation on MS-CHAP is available from Microsoft in unofficial form. The best documentation is from the Samba project, which has reverse-engineered the SMB protocol and has given Microsoft a significant number of corrections to their documentation. The version here is from Paul Leach, *CIFS Autehntication Protocol*, dated 3/28/97 and marked "do not cite", so I will not cite it.

The server sends the client a 8 byte nonce, to which the client responds with the encryption of the nonce three times, by three keys derived from the MD4 hash of the user's password. The 16-bytes MD4 hash has 5 bytes of zeros appended and then broken into three 7 byte pieces. A challenge C is presented by the server and the client responds with the response R,

$$\begin{aligned} P^h &= \mathcal{MD}4(P_u) \\ P^h_1|P^h_2|P^h_3 &= P^h|Z^5 \\ R &= \mathcal{DES}_{P^h_1}(C)|\mathcal{DES}_{P^h_2}(C)|\mathcal{DES}_{P^h_3}(C) \end{aligned}$$

The pair P^h, R is used as a shared secret for authentication and integrity checking for the remaining messages between client and server. A serial number is set to 0 for the client replay and incremented to 1 for each message sent client to server or server to client. The MAC is calculated,

$$\text{MAC}(\text{serial number}, \text{message}) = \mathcal{H}(P^h, R,\ \text{serial number},\ \text{message}\ , [U, R])\{8\}$$

and truncated to the hight order 8 bytes. Here, MD5 is the hash function. The final U, R pair is only used on the first message from client to server. All other hashes omit these elements.

2.3 MS-CHAPv2

An improved version of MS-CHAP, called version 2, was introduced. This description is taken from *Cryptanalysis of Microsoft's PPTP Authentication Extensions (MS-CHAPv2)*, by B. Schneier, Mudge and D. Wagner, September 1999.

$$\begin{array}{ccc} \text{Mudge} & \xleftarrow{SC} & \text{Gates} \\ \text{Mudge} & \xrightarrow{R,PC} & \text{Gates} \\ \text{Mudge} & \xleftarrow{A} & \text{Gates} \end{array}$$

Figure 3: MS-CHAPv2 summary

The server sends a 16-byte challenge SC. The client adds its own randomness by generating a random 16-byte Peer Authenticator Challenge PC. The two challenges are combined to an 8-byte challenge C,

$$C = \mathcal{SHA}(PC, SC, \text{client username})\{8\}$$

The first 8 bytes of the SHA-1 hash of the above information forms C. In response to the client receiving the 16-byte SC it sends the 24-byte response R and its 16-byte PC. R is formed as in MS-CHAPv1.

The protocol continues by authenticating the server. The server sends an Authenticator Response based on PC and the hashed password P^h,

$$\begin{aligned} A' &= \mathcal{SHA}(\mathcal{MD}4(P^h), R, \text{“Magic server to client constant”}) \\ A &= \mathcal{SHA}(A', C, \text{“Pad to make it do more than one iteration”}) \end{aligned}$$

The 20-bytes of A are sent as a response.

The improvements from v1 to v2 include:

1. The response is based on client randomness as well as server randomness, preventing a chosen plaintext attack.
2. The authentication is mutual, the server authenticates to the client in the last step.
3. Not shown, the LANMAN challenge response subprotocol was removed.

2.4 Bellovin-Merritt

The challenge response schemes previously described leak password information. Any server, wishing to compromise a client, has access to the input-output pair under the challenge-response function. From this, a brute force attack on the password can begin. A scheme by Bellovin and Merritt (*Encrypted key exchange: password-based protocols secure against dictionary attacks*, S. Bellovin and M. Merritt, Proceed. IEEE Comp. Soc. Symp. on Res. in Security and Privacy, May 1992, 72–84) uses a Diffie-Hellman key excange to mask the user's password.

To review Diffie-Hellman: a prime p and an generator g of F_p is made public knowledge by some trusted authority. Party B sends party M the value $g^{r_B} \bmod p$; party M sends party B the value $g^{r_M} \bmod p$, where r_B and r_M are randomly selected values, secret to B and M. Each party computes $g^{r_B r_M} \bmod p$. This value is now known to B and M but, under the D-H assumption, cannot be calculated by any other party, including a party who witnessed all other values.

Bellovin-Merritt performs D-H with the exchanged partial secrets encrypted by the shared password. They then confirm to each other that they know the shared secret.

$$
\begin{array}{rclcrcl}
r_B & \in_R & Z & & & & \\
M_1 & = & \mathcal{E}_{BM}(g^{r_B} \bmod p) & \mathcal{B} \xrightarrow{M_1} \mathcal{M} & & & \\
 & & & & r_M & \in_R & Z \\
 & & & \mathcal{B} \xleftarrow{M_2} \mathcal{M} & K & = & (g^{r_B})^{r_M} \bmod p \\
 & & & & M_2 & = & \mathcal{E}_{BM}(g^{r_M} \bmod p) \\
R_1 & \in_R & Z & & & & \\
K & = & (g^{r_M})^{r_B} \bmod p & \mathcal{B} \xrightarrow{M_3} \mathcal{M} & & & \\
M_3 & = & \mathcal{E}_K(R_1|0) & & & & \\
 & & & \mathcal{B} \xleftarrow{R_1,M_4} \mathcal{M} & R_2 & \in_R & Z \\
 & & & & M_4 & = & \mathcal{E}_K(R_2|1) \\
 & & & \mathcal{B} \xrightarrow{R_2} \mathcal{M} & & &
\end{array}
$$

Figure 4: Bellovin-Merritt

A dictionary attack can be attempted against $\mathcal{K}(R_1|0)$, for instance, but recovering $g^{r_B r_M}$ mod p tells nothing of the two factors $g^{r_B} \bmod p$ and $g^{r_M} \bmod p$, so a dictionary attack against the password $B - M$ has no place to start.

2.5 Zero-Knowledge: Fiat-Shamir

The Bellovin-Merritt removes password guessing, but both parties, Bellovin and Merritt, need to know a common secret. This means, Merritt knows Bellovin's password and can therefore impersonate Bellovin if ever their friendship goes sour. The technology of zero-knowledge is an interaction between two parties, prover and verifier, in which the prover convinces the verifier of a fact, however, the verifier learns nothing at all from the interaction, except that the fact is true. In addition, nothing can be gained either by prover or verifier by cheating, that is, by failing to implement the protocol correctly or my making misstatements.

In the context of passwords, the prover is the client, the verifier is the server. The server is convinced by the client that the client knows the password but receives no other information from the interaction except this belief. A cheating server cannot extract more information from the client than in a fair exchange; and a cheating client cannot convince the server that it knows the password.

This all proceeds in a probabilistic setting. Either party can be replaced by a random number generator which, purely by chance, completes the protocol as would a true prover or verifier. However this is true of all security, our guarantees are all bounded by the probability of a luck guess.

The Fiat-Shamir scheme is the oldest and simplest. There are several others which address the impracticality of the scheme, due to its large computational and communication overhead. We describe this scheme to make the concepts clear, and as yet I don't know of any commercial implementations, so an impractical scheme is a good as any other.

The mathematical background is the following: given two distinct primes p and q the ring of integers mod $n = pq$ has certain elements with square roots, and others without square roots. It is not known, given an element $x \in Z_n$ where $x = y^2$ how to compute y without first factoring n into p and q, and it is not known how to efficiently factor n. The asymmetry is intriguing: given an x of my choosing, choosing first a y and then presenting $x = y^2$, I can provide a square root of x, but no one else can. The Fiat-Shamir scheme provides to the server proof that I know a y for x such that $y^2 = x$ without revealing y.

A trusted authority chooses p and q and makes $n = pq$ public. Each client registers with the trusted authority a public v, where $v = s^2 \bmod n$, and the client holds s secret. The protocol is as follows:

1. *Committment:* The client sends the server a random $x = r^2 \bmod n$, keeping the randomly selected r private.
2. *Challenge:* The server asks for either the square root of x or the square root of vx.
3. *Decommittment:* The client sends either r or sr, according to the server's challenge.

This protocol is repeated t times until the server is sufficiently sure that the client knows s.

The client must always be able to correctly decommit for the server to be convinced and to authenticate the client. If the client knows s than it can always correctly decommit. If the client does not know s then it cannot possibly be able to decommit both x and vx, for if it could, the ratio of these decommittments is s, which we claim the client does not know. Since the client cannot predict the challenge, for t iterations of the protocol, it has a $1/2^t$ chance of deceiving the server. These are the guarantees of authenticity for the server.

The proof that the server learns nothing is ingenious. The server receives a stream of t random numbers. Since they are truly random, the server needn't have interacted with the client for these numbers. Rather, the server could have generated these numbers itself. It would generate its own r and send itself $r^2 \bmod n$ if it intends to ask for the square root of x, or send itself $r^2/v \bmod n$ if it intends to ask for the square root of vx. It is not possible to distinguish an x formed by the square of a random r from an x formed by the square of r divided by v, both a equally distributed random elements $x \in Z_n$ such that x has a square root mod n.

Since the entire conversation between client and server can be replaced with a conversation of the server with itself, the server cannot have learned anything which depends on the client. It has learned nothing it did not already know.

3 One-time passwords

A perfectly secure method of authentication is for the client to make use of a random list of passwords, shared with the server, and both client and server agree that each password will be used only once. A password's use renders it useless. Therefore an eavesdropper learns only useless information.

It is impractical to implement this scheme strictly. Rather than a list of truly random passwords, a pseudo-random generator replaces the list to generate new passwords as required. Breaking the pseudo-random generator breaks the one-time password scheme, so the pseudo-random generators are chosen with great care.

3.1 Lamport Hash and S/Key

The Lamport hash (*Password authentication with insecure communication*, Comm. of the ACM, Vol 24, 11, Nov. 1981, pp. 770–772) uses a strong hash function to generate the list of passwords. Starting from a client secret p, the passwords are,

$$\begin{aligned} p_0 &= \mathcal{H}(p) \\ p_i &= \mathcal{H}(p_{i-1}), \quad i > 0 \end{aligned}$$

where $\mathcal{H}$ is a suitably strong hash function. The passwords are used in order of descending i. Presented with an i, the client can quickly reconstruct p_i. The server, having in its database a p_j with $j > i$ can quickly hash forward to verify the password.

S/Key is an implementation of Lamport's scheme. The implementation problem to deal with is insuring that client and server can agree on the next i to use.

Question: How does S/Key deal with this problem? What are the issues?

3.2 Secure ID

Another class of one-time password schemes are based on a shared time and a shared secret. The hash of the secret and the time forms the password, good for a small interval of time. In truth, there is little to distinguish this from a challenge-response scheme where the challenge is the time, treating time as a random value, never again reused. Since client and server agree on the time, the time is not sent and therefore only one message is required, making the scheme superficially similar to one-time passwords.

4 Indirect Authentication

Centralized user administration gives rise to authentication servers. Sun introduced NIS (originally known as Yellow Pages) to centralize the password database, as well as many other user and system configuration files using the same mechanism. Windows uses NTLM, the concept of NT Domains, and more recently , Active Directory. In fact, it was the highly unsuccessful Novell that was the master of this sort of thing.

The first indirect authentication scheme provides access to a single machine. Key Distribution Centers solve a larger problem, sometimes called single-logon. With a KDC the user authenticates to a central authentication server and can receive permission on many different network services using the central server.

4.1 RADIUS

Remote Authentication Dial In User Service, also known as RADIUS, see RFC 2865, C. Rigney et. al., consists of a client-server architecture in which the authenticating host contacts the RADIUS server, presenting the client's information in an *Access-Request* message, and waiting for either an *Access-Accept* or *Access-Reject* reply from the RADIUS server.

The protocol is packet-oriented, with three types of messages:

1. *Access-Request*, from client to server to verify authorization information.
2. *Access-Accept/Access-Reject*, from server to client, informing client of authentication decision.
3. *Access-Challenge*, from server to client, to send challenge information to the client, and expecting a subsequent Access-Request message responding to the challenge.

A share secret authenticates the Radius server to its clients. Each Access-Request packet includes a 16-byte Authenticator, chosen at random, and unique of the lifetime of the shared secret. Response packets must contain the correct Response Authenticator,

$$\begin{array}{rcl} \text{ReqAuth} & \in_R & \{\text{16-byte}\} \\ \text{RespAuth} & = & \mathcal{MD}5(\text{response data}, \text{ReqAuth}, \text{secret}) \end{array}$$

If the Access-Request packet includes a password, the password is encrypted by exclusive-or with a 128-bit value derived from the Request Authenticator and the secret,

$$\mathcal{E}(\text{password}) = \mathcal{MD}5(\text{secret}, \text{ReqAuth}) \oplus \text{password}$$

4.2 Needham-Schroeder

The basic network logon scheme is due to (R. Needham and M. Schroeder, *Using encryption for authentication in large networks of computers*, Comm. of the ACM, Vol. 21, Dec 1978, pp. 993–999.) Once the user is authenticated to the Key Distribution Server, the user can establish privileges for many network services.

The KDC is key server, and is maintained with the strictest security. It shares with each user a secret user key. For two users to authenticate, the KDC will generate a fresh session key and send the key to both parties, encrypted with the user's secret key. The users then mutual authenticate by demonstrating knowledge of the fresh session key. For engineering reasons, the KDC communicates with only one of the parties, the party requesting the connection. That party will forward KDC messages for the second party on behalf of the KDC.

$$
\begin{array}{lcl}
\begin{array}{rcl} R_1 & \in_R & Z \\ M_1 & = & R_1, \mathcal{N}, \mathcal{S} \end{array} & \mathcal{N} \xrightarrow{M_1} \mathcal{KDC} & \\
 & \mathcal{N} \xleftarrow{M_2} \mathcal{KDC} & \begin{array}{rcl} K_{NS} & \in_R & Z \\ T & = & \mathcal{E}_S(K_{NS}, \mathcal{N}) \\ M_2 & = & \mathcal{E}_N(R_1, \mathcal{S}, K_{NS}, T) \end{array} \\
\begin{array}{rcl} R_2 & \in_R & Z \\ M_3 & = & \mathcal{E}_{K_{NS}}(R_2) \end{array} & \mathcal{N} \xrightarrow{T, M_3} \mathcal{S} & \\
 & \mathcal{N} \xleftarrow{M_4} \mathcal{S} & \begin{array}{rcl} R_3 & \in_R & Z \\ M_4 & = & \mathcal{E}_{K_{NS}}(R_2 - 1, R_3) \end{array} \\
M_5 = \mathcal{E}_{K_{NS}}(R_3 - 1) & \mathcal{N} \xrightarrow{M_5} \mathcal{S} &
\end{array}
$$

Figure 5: Needham-Schroeder

4.3 BAN Logic and the failure of Needham-Schroeder

There is considerable concern over the ad hoc nature of reasoning about security protocols. BAN logic attempts to formalize this reasoning. (*A logic of authentication*, M. Burrows, M. Abadi, R. Needham, SRC Report 39, 1990) This system has itself been criticized (find reference), for instance, in the incorrect assumption that confidentiality (encryption) implies authenticity. However, it has discovered a flaw in Needham-Schroeder.

BAN logic captures to protocol flaw of replay in the formalization of *freshness*. Suppose Alice says X to Bob. Bob then believes that Alice once believed X. If X is new or novel, then Bob will believe that Alice currently believes X. A nonce provides the necessary freshness to transform what Alice once said, for example, that this password was sufficient to authenticate, into what Alice still believes, that this password is sufficient to authenticate.

There is nothing in Needham-Schroeder to guarantee freshness of the ticket generated by the KDC. While the KDC will not create a ticket unless the KDC believes the encrypting key is sufficient for authentication, the ticket receiver has no reason to believe that the ticket was created recently, that is, that the KDC still believes that the encrypting key is good. Needham can bluff Schroeder into a conversation if ever it has been allowed to converse with Schroeder, unless Schroeder's own key is revoked.

4.4 Otway-Rees

Another protocol, similar to Needham-Schroeder is Otway-Rees. (D. Otway and O. Rees, *Efficient and timely authentication*, Operating systems review, Vol 21, No. 1, Jan. 1987, p 7.)

$$\begin{array}{rcl} N_a, N_o & \in_R & Z \\ M_1 & = & \mathcal{E}_o(N_a, N_o, \mathcal{O}, \mathcal{R}) \end{array} \qquad \mathcal{O} \overset{N_a,\mathcal{O},\mathcal{R},M_1}{\longrightarrow} \mathcal{R}$$

$$\mathcal{KDC} \overset{M_1,M_2}{\longleftarrow} \mathcal{R} \qquad \begin{array}{rcl} N_r & \in_R & Z \\ M_1 & = & \mathcal{E}_r(N_a, N_r, \mathcal{O}, \mathcal{R}) \end{array}$$

$$\begin{array}{rcl} K & \in_R & Z \\ M_3 & = & \mathcal{E}_r(N_r, K) \\ M_4 & = & \mathcal{E}_o(N_o, K) \end{array} \qquad \mathcal{KDC} \overset{N_a,M_3,M_4}{\longrightarrow} \mathcal{R}$$

$$\mathcal{O} \overset{M_4}{\longleftarrow} \mathcal{R}$$

Figure 6: Otway-Rees

Question: find the original protocol and apply BAN suggestions to a modified version.

4.5 Kerberos

Kerberos is a network authentication protocol based on Needham-Schroeder. It has three main differences from Needham-Schroeder. First, the mutual authentication in the final steps of N-S is done using timestamps, rather than random nonces, and is accomplished in two steps rather than three. Second, the conversations between a client and the KDC begin with the assignment of a session key, which replaces the client key for the remainder of the session. This is done to reduce dictionary attacks on the shared key. It is expected that the shared key will be user derivable and therefore of small entropy. Third, so that the KDC does not itself have to remember the session key, a Ticket-Granting-Ticket is given to the client when the session key is generated. This TGT is for the KDC's benefit. Inside the TGT, encrypted by a private key to the KDC, is the session key. The KDC does not remember the session key, rather it requires the client to present the TGT. The KDC decrypts the TGT and is thereby reminded of the session key.

4.5.1 Authentication Server Request/Reply

The Kerberos protocol can be neatly broken down into three classes of Request-Reply. When the user first logs in, the user's workstation and the KDC authenticate the user, assign a session key, and create a TGT.

The request is simply the name of the user, Alice.

$$\text{AS-REQ} = \text{Alice}$$

The response is,

$$\begin{aligned} \text{TGT} &= \mathcal{E}_X(\mathcal{A}, S_A) \\ \text{AS-REP} &= \mathcal{E}_A(S_A, \text{TGT}) \end{aligned}$$

where X is the super-secret KDC key, known only to the KDC.

4.5.2 Ticket Granting Service Request/Reply

Alice asks the KDC for a shared key with Bob. It presents the request with its TGT and an authenticator. The authenticator is the time encrypted with the session key. The KDC verifies the information, creates the key K_{AB} for Alice and Bob and returns it to alice as an encrypted (by the session key) packet including K_{AB} and K_{AB} encrypted by Bob's key, the ticket to forward to Bob.

The request is,

$$\text{TGS-REQ} = \text{Alice-Bob}, \text{TGT}, \mathcal{E}_{S_A}(t)$$

The reply is,

$$\begin{aligned} T_{AB} &= \mathcal{E}_B(\mathcal{A}, K_{AB}) \\ \text{TGS-REP} &= \mathcal{E}_{S_A}(\mathcal{B}, K_{AB}, T_{AB}) \end{aligned}$$

4.5.3 Application Request/Reply

Alice sends Bob the ticket, and an authenticator, the time encrypted by K_{AB}. Bob responds with the time sent plus one encrypted by K_{AB}.

Alice tells Bob,

$$\text{AP-REQ} = T_{AB}, \mathcal{E}_{K_{AB}}(t)$$

and Bob responds,

$$\text{AP-REP} = \mathcal{E}_{K_{AB}}(t+1)$$

5 Appendices

5.1 DES, 3DES, DESX

The Data Encryption Standards DES and 3DES are described in FIPS 46-3. First adopted in 1977, DES has withstood attack up to this day. Its 56-bit key length, however, is too short to withstand brute force attack, see the EFF's book on the subject. 3DES and DESX extend the useful life of DES by introducing more key bits.

DES is a 64-bit block cipher with 56-bit keys. The space of inputs, $[0, 2^{64} - 1]$ is mapped back onto itself in an invertible manner (a permutation) according to one of 2^{56} patterns, given the key. DES runs in two modes, applying the same key to produce the permutation or the inverse permutation, corresponding to encryption mode or decryption mode, respectively.

We need to describe precisely the security of DES. There are many ways to do this, but the following is sufficient. Suppose we are given a number of pairs (x_i, y_i) related by $y_i = \mathrm{DES}_k(x_i)$. That is, y_i is the encryption of x_i using key k. What is the most efficient algorithm for finding k given these pairs? At present, the best approach is to select one pair (x_o, y_o) and scan all keys k looking for the value k_o such that $y_o = \mathrm{DES}_{k_o}(x_o)$. Since only this brute force method exists, DES is considered unbroken. However there are not that many keys to try, so the brute force method is effective.

To continue the usefulness of DES, triple DES uses three keys, k_1, k_2, k_3 and encrypts the input three times, first with k_1, then with k_2, then with k_3. In addition, the second encryption runs DES in decryption mode.

$$3\mathrm{DES}_{k_1,k_2,k_3}(x) = \mathrm{DES}_{k_3}(\mathrm{DES}^{-1}_{k_2}(\mathrm{DES}_{k_1}(x)))$$

There are three modes of operation allowed: all three keys independent, all keys the same, and $k_1 = k_3$ with k_2 independent. There are two motivations for these patterns of usage. The mode $k_1 = k_2 = k_3$ reduces 3DES to DES, so hardware supporting 3DES trivially supports DES. The mode $k_1 = k_3$, with k_2 independent, is interesting since we can show that three independent keys give no more security than two. In general, cascading an encryption with n keys only gives the security of $2\lfloor(n+1)/2\rfloor$ keys. Hence there is no point in using three independent keys over two independent keys.

Here is the proof that n keys give no more security than $\lfloor(n+1)/2\rfloor$ keys. Divide n as evenly as possible into n_1 and n_2. If n is even, $n_1 = n_2 = \lfloor(n+1)/2\rfloor$, else $n_1 = n_2 + 1 = \lfloor(n+1)/2\rfloor$. Given the pair (x, y) make a table of n_2-$\mathrm{DES}^{-1}_{k_{n_1+1},\ldots,k_n}(y)$ as the keys $k_{n_1+1}, \ldots, k_n$ take on all values. Then try to match a value with n_1-$\mathrm{DES}_{k_1,\ldots,k_{n_1}}(x)$ as the keys $k_1, \ldots, k_{n_1}$ take on all values. A match provides n keys such that n-$\mathrm{DES}_{k_1,\ldots,k_n}(x) = y$. Try additional (x, y) pairs to verify the keys. If additional pairs don't work out, continue the search.

Although a large table of values is required (2^{64} 64-bit blocks), by attacking the encryption half-forward and half-backward we search the key spaces $k_1, \ldots, k_{n_1}$ and $k_{n_1+1}, \ldots, k_n$ separately, with time at most twice the time to search the larger key space.

DESX is another approach to extending the usefulness of DES. Three keys are used, k, k_1 and k_2, where k is 56-bits, k_1 and k_2 are 64-bits.

$$\mathrm{DESX}_{k,k_1,k_2}(x) = k_1 \oplus \mathrm{DES}_k(k_2 \oplus x)$$

Question: is there an argument against $k_1 = k_2$*?*

5.2 Hash functions: MD4, MD5 and SHA

MD4 and MD5 are property of RSA Security and have been made publicly available for any use. A description of MD4 can be found in RFC 1186, and of MD5 in RFC 1321.

A hash function if a function which is easy to compute but hard to invert. Compared to an encryption, a hash function cannot be uniquely inverted since there are less bits in the output than in the input, therefore there are many inputs which can give the same output. Even so, it is hard to compute even a single possible input for an output. Furthermore, it is hard even to compute to inputs which give the same output, where the output is not specified before-hand. This is called *collision resistance.*

The statistics of collision resistance differs from that of inversion. Calling our hash function $\mathcal{H}$, finding an x given y such that $y = \mathcal{H}(x)$ requires on average $1/|Y|$ trials, where $|Y|$ is the size of the space of outputs. MD4 and MD5 have 512 bit input spaces and 128 bit output spaces. Hence we expect to find such an x in 2^{128} trials. However, finding different x_1 and x_2 such that $\mathcal{H}(x_1) = \mathcal{H}(x_2)$ requires only 2^{64} trials, as we argue: On the first 2^{64} calculations of $\mathcal{H}(x_1)$ we assume pessimistically that there are no collisions. Then the fraction $1/2^{64}$ of the output space is occupied. Therefore we expect a collision within the next 2^{64} values of x_1 attempted.

A hash function is secure if the statistical strength, as described above, is also the best known emperical strength, that is, no better attack is known.

Hash functions constructions allow hashing of arbitrary length bit strings into the output space. On construction is to apply the function iteratively on a padded input which is a multiple of 512 bits. The initial state is set to a known constant K and each next 512 bits of input provides the state for the next 512 bits of input,

$$\begin{aligned} h_0 &= \mathcal{H}(m_0, K) \\ h_i &= \mathcal{H}(m_{i-1}, h_{i-1}), \ \ i > 0 \end{aligned}$$

MD4 and MD5 prescribe mandatory padding of the input to an even multiple of 512 bits which includes appending the original length of the input to the input as part of the padding.

MD4 has been broken by H. Dobbertin, *Cryptanalysis of MD4*, 3-ird Fast Soft. Encry., LNCS 1039, Springer-Verlag, 1966, 53–69. See also J. of Cryptology.

SHA-1 is a modified MD5 proposed by NIST for Federal Government applications. See FIPS 180-1. It uses 5 32-bits words of internal state, compared to MD5's 4 32-bit words. That is, the output is 160-bits rather than128-bits. It still works on 512-bit inputs.

6 References

R. Morris and K. Thompson, *Password Security: a case history*, Comm. of the ACM, Vol. 22, Nov. 1979, 594–597.

H. Dobbertin, *Cryptanalysis of MD4*, 3-ird Fast Soft. Encry., LNCS 1039, Springer-Verlag, 1966, 53–69. See also J. of Cryptology.

R. Needham and M. Schroeder, *Using encryption for authentication in large networks of computers*, Comm. of the ACM, Vol. 21, Dec 1978, pp. 993–999.

L. Lamport, *Password authentication with insecure communication*, Comm. of the ACM, Vol 24, 11, Nov. 1981, pp. 770–772.

A. Fiat and A. Shamir, *How to prove yourself: practical solutions to identification and signature problems*, CRYPTO '86 (LNCS 263), 1987, pp. 186–194.

RFC 2865 Remote Authentication Dial In User Service (RADIUS) C. Rigney et. al.

M. Burrows, M. Abadi, R. Needham, *A logic of authentication*, SRC Report 39, 1990.

D. Otway and O. Rees, *Efficient and timely authentication*, Operating systems review, Vol 21, No. 1, Jan. 1987, p 7.

A description of MD4 can be found in RFC 1186, and of MD5 in RFC 1321.

Data Encryption Standards (DES), FIPS PUB 46-3, (1999).

Secure Hash Standard, FIPS PUB 180-1 (1995).

Password Usage, FIPS PUB 112 (1985).