Review

Last time, we looked at SHA-1 and its variants in detail. You have a homework problem that works with a similar algorithm.

I promised to look up the details on the weaknesses in MD5. There are many – lots of people have been analyzing for a long time – but the most devastating is the ability to generate two postscript files with the same MD5 value. This is described at http://www.cits.rub.de/MD5Collisions/. The trick here is that postscript is a programming language, so one has the ability to alter the message “online” to make the MD5 hash the same.

A harder to understand attack was developed by Xiaoyun Wang and Hongbo Yu [“How to Break MD5 and Other Hash Functions”. EUROCRYPT 2005]. Their attack applies to many hashes, although not to SHA-*.

Since MD5 has a lot of history, there are documents whose checksum or digest is the MD5 to use two digests for any message. The feeling is that while it may be easy to generate a collision with hash 1, it is unlikely that the same two messages collide under hash 2. For example, a recent post on a patch to fix an implementation vulnerability of RSA in SSL included pointers to documents like this:

openssl-0.9.8c.tar.gz
MD5 checksum: 78454bec556bcb4c45129428a766c886
SHA1 checksum: d0798e5c7c4509d96224136198fa44f7f90e001d

Preview of the Rest of the Semester

The rest of the semester will be more about security than about cryptography. Naturally, neither of these is much use without the other. The security material is probably of more interest to system administrators, while cryptography is of more interest to academic computer scientists. But neither one of these is much use without the other, either.

The major topics will be:

Authentication and Certification [Ch. 14]  
Email security, PGP [Ch. 15]  
Internet Protocol Security [Ch. 16]  
Web Security, SSL, TSL [Ch. 17]

Passwords
The most familiar form of authentication is the password. The traditional UNIX password scheme, as described on pp. 582ff of Stallings, goes as follows:

1. User types a password. A 12-bt salt value is computed when the password is generated;
2. The null word is encrypted with a DES modification that depends on the salt, using the password as the key;
3. Repeat the encryption 25 times.

The salt increases the size of the key space by a factor of 4096.

This process is repeated when a user starts login, and the result is compared with the stored result.

This scheme is very old, and other UNIX systems, including LINUX, have more sophisticated password schemes.

Authentication and Kerberos

Some feel that authentication is more important than confidentiality. I really do not care if someone overhears me withdrawing $100 from the bank, but I do care that they make sure that it really is me. I do not think that the usual “security words” (you know, e.g. mother’s maiden name) are very secure (too many people overhear it), so I have insisted that they authenticate me by photo ID. This was awkward for a while, since they were insisting on photo ID for deposits, but anyone who wants to impersonate me for the purpose of making deposits to my account is welcome to do so.

Plain encryption does not offer much help in this situation; an eavesdropper who decodes my cryptic withdrawal may be able to replay it and make a withdrawal. The authentication is much more important.

While Stallings describes Kerberos Version 4, Kerberos is no longer supporting that version, so we will concentrate on Version 5. The Kerberos website http://web.mit.edu/Kerberos describes the vulnerabilities of Kerberos 4. A major one is its reliance on DES, but there were other design issues as well. The paper “The Perils of UNauthenticated Encryption: Kerberos Version 4,” by Tom Yu, Sam Hartman, and Kenneth Raeburn, is available at the Kerberos website.

The basic idea of Kerberos is that of an authentication server AS. The most naîve approach, as outlined on p. 404 of Stallings, involves a client $C$ requesting a Ticket $T$ allowing access to a server $V$. Here, $ID_C$ identifies the client, $ID_V$ identifies the server, $P_C$ is the client’s password.

\[
\begin{align*}
C \text{ to } AS & : & ID_C || P_C || ID_V \\
AS \text{ to } C & : & T \\
C \text{ to } V & : & ID_C || T
\end{align*}
\]

The ticket $T$ is $ID_C || AD_C || ID_V$ encrypted with the server’s key $K$, where $AD_C$ is the client’s network address.
How does this work? First, the use of the password $P_C$ authenticates the client, so the authentication server is comfortable about the client’s identity. The use of the server’s key means that only the server can decrypt the ticket to see that it is genuine. Furthermore, the inclusion of the network address of the client in the ticket prevents a replay attack from a different address.

The biggest difficulty here is that the password is sent in the clear, so an eavesdropper can capture it.

A more sophisticated scheme involves a Ticket Granting Server.