Passwords and Authentication

Justin Hibbits

April 26, 2004

Abstract

Computer security has been an issue since the first inception of the UNIX opearting system. Passwords were the first method of authentication, but are no longer the only method. Now there exist other methods including smartcards, voice, or biometrics, and they allow authentication against more than just the local computer or the local network and internet, such as databases and other applications. Password storage and authentication methods are increasingly important, yet most current implementations of storage are very weak, especially since one implementation of Windows passwords is optionally reversible.

Introduction

In today's world, security is a very big concern, and passwords are the primary method of authentication for computer systems. Strong passwords are a must, as are strong encryption algorithms to store those passwords, since they must ultimately be stored somewhere, in order for a user to authenticate to a computer system. This document analyzes the Windows NT implementation of passwords, both native and LAN Manager (Windows 95 compatible), Linux and UNIX passwords, including MD5, DES, Blowfish, and Shadow password files, and the authentication methods of each system.

Operating System Password Schemes

Operating systems use different methods for authentication and password storage. Linux and UNIX use DES, MD5, and Blowfish for password hashes, along with Shadow password files to hide the hashes. Windows NT uses MD4 and DES for encrypting passwords, and a file called the Security Accounts Manager, a binary file, to store the password hashes, and other account information.

Linux, UNIX, and Shadow Passwords

Since its creation, UNIX and UNIX-like systems have used a variety of password encryption methods, primarily DES. However, due to the weaknesses in the DES algorithm, other

algorithms have been used, including MD5 and Blowfish.

DES and its Weaknesses

According to Robert Morris, the original author of the crypt(3) function, the original algorithm used in crypt(3) was far too fast, so a slower algorithm needed to be created. This was DES, standardized by the National Bureau of Standards (now the National Institute of Standards and Technology). and designed to be slow when implemented in software.

The Data Encryption Standard was first developed in 1988 and later revised in 1993 to fix some weaknesses. UNIX and UNIX-like systems use a variation of the DES algorithm with extensions to discourage hardware implementations of key searches.

The DES algorithm used in the crypt(3) function on many UNIX-like systems, including Linux and FreeBSD, works as follows:

1. A random 2 character (12 bits, as only the lower 6 bits are used) is chosen to hash the password.

2. The password is truncated to 8 characters, and the lower 7 bits of each character, since the algorithm is only a 56-bit hash.

3. The password itself is perturbed in one of 4096 ways (using the salt), following the DES algorithm.

4. This string is used to repeatedly encrypt a constant string, usually all NUL characters, but may be different. In one implementation, it encrypts the string 25 times.

5. The resulting string is repacked into 11 printable characters, and stored in the password file, usually /etc/passwd.

As the listing suggests, DES encrypted passwords only allow a maximum of 8 characters in the password. For this reason, the MD5 digest algorithm replaced the DES algorithm for many implementations, as it allows a maximum of 127 characters in the password, much stronger than DES passwords. Additionally, DES encrypted passwords have a keyspace of only 2^{56}, or $7.2 * 10^{16}$, which makes exhaustive searches of the namespace feasible using massively parallel systems.

Although DES is slow in software, there are several chips that are commercially available which perform DES encryption. These chips are as much as three orders of magnitude faster than software implementations, but are very expensive. Additionally, to slow down the chips, the internal tables are wired to depend on the random salt. The only thing preventing an attacker from using one of these chips is the unthinkable cost associated with a custom chip, since commercial chips are crippled.

MD5 and Shadow Passwords

MD5 is an extension to the cryptographically weak MD4 algorithm. Among other additions, it adds a fourth iteration to the algorithm, to decrease its vulnerability to two-round attacks. However, it is still not proven cryptographically strong. In fact, it may be subject to the two-round attack against the MD4 algorithm, or at least a variation of it. However, it is still more secure than MD4, although to what degree is unknown. The MD5 algorithm works as follows (from [10]):

1. The password is padded to a length congruent to 448 modulo 512 (also -64 mod 512). The padding is performed as follows: A single "1" bit is appended, followed by zero or more "0" bits, until it is congruent to 448 mod 512. At least 1 bit, and at most 512 bits are appended.

2. A 64-bit representation of the length of the password before padding is appended to the padded password.

3. Four 4-word registers are initialized with the following values to compute the message digest:

   - Word A: 01 23 45 67
   - Word B: 89 ab cd ef
   - Word C: fe dc ba 98
   - Word D: 76 54 32 10

4. The password is encrypted as in Appendix A.

5. The password is output in the registers (A,B,C,D). This password is generally stored in the password file as 1<string>$, where <string> consists of an 8-byte salt (as opposed to the 2-byte salt of DES), followed by a 22 character string from the characters [a-zA-Z0-9./].

MD5 passwords are used on Linux, BSD, Solaris, and probably most other UNIX and UNIX-like systems, although most systems still default to DES for backwards compatibility. This may also be by design, to ensure that system administrators understand the system that they are deploying, and lock it down themselves, instead of blind deployment, which is done in many corporate settings.

Linux and other UNIX-like operating systems, such as Solaris and Mac OS version 10.3 use *shadow passwords*. Shadow passwords hide the actual password in a separate file, so as to be unviewable by normal users, and placing all the relavent data into the global password file. This augments the security of MD5 password hashes, by preventing users from reading the hash. Shadow passwords improve security of passwords greatly, since most login systems allow only 3 login attempts before locking the user out for a period of time. This prevents

brute-force attacks, by allowing time for the system administrator to lock a user's account before the password is cracked.

MD5 is considered strong enough for most cases, however it is recommended to use SHA-1, as it is 160-bit encryption, and has been found to be collision resistant. Nothing can be completely collision proof. Blowfish is another very strong algorithm, and has been uncrackable as of yet.

Blowfish

Blowfish is a relatively new algorithm developed by Bruce Schneier in 1993. It is a secret key cipher, with the block size being 64-bits, and the key size being any length up to 448 bits. From [1], "data encryption occurs via a 16-round Feistel network." Each round consists of a key permutation and a key and data substitution. All operations are on 32 bit words, and consist only of XOR and addition operations, along with four indexed array lookups during each round. [8]

Blowfish uses 18 32-bit subkeys, which must be precomputed before any encryption. These subkeys are generated in this case from the user's password. These subkeys are labelled $P_1, ...P_18$ and are collectively known as the *P Array* ([8]).

The encryption of the Blowfish algorithm is done as follows:

1. Each 64-bit block is divided into two 32-bit blocks, labelled L_0 and R_0.

2. L_0 is XORed against P_1, and the result is XORed against R_0. The two halves are then swapped. This is repeated 15 times, with keys $P_2 - P_{16}$, using L_{i-1} and R_{i-1}, from $2 \leq i \leq 16$. This process uses four arrays called *Substitution Boxes*, or *S-boxes*.

3. After 16 iterations, the halves are swapped again, undoing the last swap, and each half is XORed with another key, as follows:

   $R_{17} = L_{16} \oplus P_{17}$

   $L_{17} = R_{16} \oplus P_{18}$

EksBlowfish uses Blowfish for actual encryption, only the subkey generation algorithm differs. The EksBlowfish algorithm initializes the subkey generation using parameters for *cost*, *salt*, and *key*. The *cost* parameter "controls how expensive the key schedule is to compute." ([8]). The *salt* is a 128-bit random number, used so that the same key need not generate the same hash. Finally, the *key* is the user's password.

The keys are generated by first copying the digits of π into the subkeys, then into the S-boxes. Then the P-Array and S-boxes are modified using the 128-bit salt, and the key. First, the P-Array is XORed with the key, XORing the first 32 bits of the key with P_1, the second 32-bits with P_2, and so on. If the key runs out of bits, it cycles back to the beginning, and continues to XOR with the subkeys.

The first 64 bits of the salt are then blowfish encrypted using the current state of the key. The result replaces P_1 and P_2. The result is also XORed with the second 64 bits of the

salt, and this result is then encrypted with blowfish, replacing subkeys P_3 and P_4. This is then XORed with the first 64 bits of the salt, and replaces P_5 and P_6. The process continues until all 18 subkeys are replaced, and the S-boxes.

This final set of subkeys is then used to encrypt a constant string. In OpenBSD, the string is "OrpheanBeholderScryDoubt". The password is then stored in /etc/passwd. Since the Blowfish algorithm has not been cracked, some may consider it safe to keep in /etc/passwd, even though it is readable by everyone.

Cracking UNIX Passwords

The Shadow password suite was designed to prevent people from getting a hold of the password listing, since given the password listing, one can crack a large number of passwords, since it is statistically proven that many people use dictionary passwords. There are two popular UNIX password crackers on the internet today: Crack and John the Ripper, both using compiled dictionaries as a crack reference. Crack was one of the earliest, and still powerful crackers. However, John the Ripper is more popular, as it can enumerate dictionary entries into so-called "leet-speak" and subtle misspellings.

Dictionary crackers, although very useful, may be against institution policy, or, in some cases, even illegal. Some system administrators have even been fired and/or put in jail for using dictionary crackers for auditing purposes. Anyone wishing to perform even light password checking should do so with the consent of his/her boss, to make sure it complies with company policy. Because of this, some password suites, such as PAM, later discussed, use libcrack, a password cracking library and weak-password verifier, to verify the strength of passwords, and prevent the unintentional, or even intentional, use of weak passwords. Some network policies even force constraints on passwords, such as requiring punctuation as well as alphanumeric (both upper and lowercase) and requiring passwords to have a certain minimum length, the most popular being 8 characters minimum.

Windows NT

There is no official documentation available for the password hash database format, so the encryption can not be determined. However, many people have reverse engineered the format. From reference [3], it is shown that Windows NT4 uses a rather insecure algorithm for encrypting user passwords, as follows:

1. Convert the password to unicode.
2. Make an MD4 hash out of the unicode password.
3. Encrypt the hash with DES, using the user ID as the crypt key. This is for obfuscation purposes only.

The insecurity of this encryption is due to the weakness of the MD4 digest algorithm. The MD4 algorithm is not collision free, and is succeptible to a two-round attack. MD5 is a

much stronger hash function based on MD4, however still vulnerable to a two-round attack, as shown in [5]. Windows NT also uses LANMAN passwords for Windows 95/98 computer access. This encryption algorithm is much less secure:

1. Convert password to uppercase, remove illegal characters, and pad to 14 characters, truncating if the password is longer, padding with 0 if it is shorter. The Windows NT GUI does not allow passwords longer than 14 characters, so this is not an issue.

2. A known string is DES encrypted using the first 7 characters of the password as the key, and another string is DES encrypted using the last 7.

3. The two DES hashes are concatenated to form a 16 byte string.

4. The resulting hash is then encrypted with DES, using the user ID as the crypt key. This, as with the NT password encryption, is for obfuscation purposes only.

Cryptanalysis of Windows NT Passwords

Since Windows NT passwords are stored in two formats, it makes logical sense to attack the weaker format, which is the LAN Manager format. The LAN Manager format is weak to maintain compatibility with Windows 95/98/ME operating systems, which, by design, have weak passwords.

Since the LAN Manager password encryption algorithm converts all characters to uppercase, 26 characters are lost, reducing the message space. Additionally, since it must be represented by the OEM character set, it restricts it to under 256 characters - the 8-bit OEM character set. Additionally, since most keyboards do not support the entire OEM character set, the password is reduced to 66 possible characters. So the number of possible password combinations becomes 66^14, or approximately $3 * 10^{25}$, which can not be brute-forced by an exhaustive search of the valid keyspace. However, most passwords can be cracked within 20 seconds by various password crackers. The NT password is based on the Unicode character set, so does not have this restriction. However, NT goes to great lengths to enforce compatibility between the LAN Manager and NT password lists. This is the Windows NT's greatest weakness, since if one password is cracked, both are.

Second, the password is converted to two strings of 7 characters each, after being padded with NULL(0) characters. Each string is used to encrypt a constant using DES. This makes it easy to determine if the password length is below 8 characters. These two facts make the LAN Manager password the first attacked in brute-force attacks.

However, the LAN Manager password may not be available for a variety of reasons. According to the Knowledge Base article, reference [3], Windows NT may have only one of the two passwords for any of the following reasons:

- If the user account was ported from the Windows 95/98 LAN Manager, only the LAN Manager password would exist.

- If the password was changed from a Windows for Workgroups client, only the LAN Manager password would exist.
- If the password would be invalid in the LAN Manager (i.e. it has more than 14 characters, and/or contains characters not representable by the OEM character set), only the Windows NT password would exist.

When the LAN Manager password is unavailable, the NT password can be cracked. From the knowledge base article, the first encryption of the NT password uses the MD4 digest algorithm, "generally considered non-decryptable" ([3]). However, the algorithm can be broken using a two-round attack. To retrieve this encryption, the second encryption must be broken. The second password encryption can be decrypted by "anyone who has access to the double-encrypted password, the user's RID, and the algorithm." The algorithm was determined through reverse engineering to be DES. The double-encrypted password hash can be retrieved from a backup of the SAM file, which by default is readable by everyone, and the user's RID can be considered public knowledge.

Additionally, Windows 2000/XP/2003 allow for password hashes to be reversible, as required by some programs, including Microsoft's Internet Information Services (IIS), which was vulnerable to buffer overflow attacks at version 4, and may still be vulnerable. This requirement for IIS greatly decreases the security of a system. Hence, other servers should be used instead, such as Apache, which does not have that requirement. On a similar note, Microsoft has pledged its focus on security as their top priority, however this shows that they are not serious, by leaving this large hole in their own systems. Since a requirement to maintaining password security is using a one-way function, Microsoft violates this requirement for their own tools, and even state on their own webpage that storing passwords using reversible encryption is like storing them in plain text ([4]).

MD4 Two-Round Attack

MD4 is a three-round compression function. However, there were two attacks that demonstrated the weakness of this algorithm. The first attack was against the first two rounds, by Merkle, and was unpublished. The second attack was against the last two rounds, and was published by den Boer and Bosselaers ([5]).

The attack against the second two rounds of MD4, as defined in [5], works on the knowledge that it is possible that for a given input two different message blocks hash to the same output value. The observation that makes this true is that the 8 words used in the intermediate steps of rounds 2 and 3 are identical. The same observation can be made for the first 4 and the last 4 steps of each round. Consequently, the latter 8 words are given the same value in both message blocks. The two outputs then differ only in the remaining 8 words. The alternatives for these 8 words can be carefully chosen to have different values after the first 8 intermediate steps of each round, but the same value after the final 8 steps. This proves that MD4 is not collision free, and may not even be collision resistant, if the input messages can be chosen easily.

Because of the two attacks, the MD5 algorithm was designed as an extension, which basically added a fourth iteration to the MD4 algorithm. However, MD5 is not collision free either, but is more collision resistant than MD4.

Windows Dictionary Cracks

The most widely used dictionary cracker for Windows NT/2000 is L0phtCrack, a commercial cracker from @Stake, a Windows security firm. It is commercial, and costs $350 per license. However, for companies with a lot invested in Windows, it may be worth the expense for each license. John the Ripper, although very popular on UNIX systems, also works with Windows NT LAN Manager passwords. As it is free of charge, it may be a more appealing option for smaller businesses and home users.

Authentication Methods of Windows NT and UNIX Systems

Passwords are only part of the authentication process, albeit the most vulnerable. Different operating systems, and applications, use different authentication methods for protecting resources. Windows NT uses a challenge/response authentication for user authentication, with or without passwords. UNIX and UNIX-like systems use traditional password hash logins. However, this is being phased out in favor of, or in conjunction with, Sun Microsystem's Pluggable Authentication Modules, which allow authentication for applications using different authentication methods, each contained in a separate library, or module. Pluggable Authentication Modules are a relatively new technology, originating in the early 1990's, but has become widely adopted by many UNIX and Linux vendors.

Windows NT

Challenge/Response Network Authentication

Windows NT uses a challenge/response method for network and local logins, called NTLM (NT LAN Manager). This name was chosen to distinguish it from the less secure predecessor, the Windows LAN Manager (see [2]). The Microsoft Kerberos package, not compatible with the standard Kerberos protocol, is available on Windows 2000 and later, although NTLM is still available, and required if any system on the network runs Windows NT 4.0 or earlier. NTLM uses credentials for logins, which it obtains during interactive logins, and and consist of a domain name, user name, and a one-way hash (MD4) of the user's password. Instead of sending the user's password over the network, or even between processes, the NTLM uses a challenge/response protocol. This encrypts a random 'challenge' key with the user's password to prove it has access to the credentials. The challenge/response protocol behaves as follows:

1. **Interactive login only** - A user provides a username, password, and domain name. The client computes the hash of the password using MD4.

2. The client sends the plain-text user name to the server.

3. The server generates a 16-byte (128-bit) random number, the *challenge*, and sends it to the client.

4. The client encrypts the hash with the password hash, and returns the result, as the *response*.

5. The server sends the username, challenge, and response to the domain controller.

6. The domain controller locates the user's password hash in the SAM database, using the user name as the key, and encrypts the challenge using the password hash from the database.

7. Finally the domain controller compares the encrypted challenge it computed, with the response sent to it by the server (and computed by the client). If they are identical, the authentication is successful.

As one can see, this is a rather complex protocol, and most other operating systems simplify this protocol by omitting the domain controller, so that all authentication is between the client and server only.

Pluggable Authentication Modules, and UNIX Login

The UNIX login facility uses a standard login program, which requests the username and password, encrypt the password using the appropriate method, and comparing the hash against the stored hash. This is a very simple method, and has worked well since the creation of the first UNIX operating system in the early 1970's. However, more authentication methods have prompted other libraries and packages to be developed to allow the use of these methods, such as SmartCards and biometric scanners.

One such library is the Pluggable Authentication Modules, developed by Sun Microsystems. Packages are available for Solaris, Linux, BSD, and MacOS OSX. PAM allows authentication using, among others, Kerberos, LDAP, SecureID, normal passwords, shadow passwords, SMB, and NetWare. Each authentication method uses a different procedure for authentication, but they all follow the following basic pattern:

1. The user provides a username, and authentication information (password, SmartCard, etc.) to the client.

2. The client loads the appropriate PAM module, and uses it to authenticate against a server or the local machine, using the provided information.

3. If all authentication information is valid, the user is authenticated, and logs in.

Of course, there is more detail for each step, but it is module specific, but this is the general method. For example, a biometric scanner module might require both a password and a fingerprint, a fingerprint and a voice print, or just a fingerprint. A smartcard module might require the smartcard, username, and password for more security, or just the username and smartcard, or it may not require the username at all. Additionally, any application can make use of any of these technologies. PAM is a very flexible system, and is used in more than just authentication. The OpenOffice.org uses it for encrypting and password protecting documents. MySQL can use it for authentication against databases. Other applications can also make use of it.

Conclusion

Presented were password and authentication methods for UNIX, and its clones, and Windows NT. Password authentication methods rely on two main principles: that the encryption algorithm is one-way, and that the password is difficult to guess. Both operating system classes have their strengths and weaknesses for the former. However, it is impossible to control the latter, because people find it difficult to remember difficult-to-guess passwords.

The current state of password storage is enough for now, although may not be enough in the future. The Blowfish algorithm, having been proven uncrackable for the 10 years of its lifetime, seems to be the most secure hash algorithm for passwords, and will remain so for the forseeable future. MD4 is a provably weak algorithm, susceptable to a two-round attack, and MD5 suffers from this weakness as well. Authentication methods are currently very good, but may not be enough for too much longer, as computers get more powerful. A possible replacement to the MD5 hash is the SHA-1 algorithm, which, as previously stated, offers 160 bits of encryption. However, Blowfish is a currently used algorithm on FreeBSD and OpeNBSD, very secure, and can very easily be ported to other UNIX-like operating systems. This makes it a very powerful choice for future directions of password storage.

A MD5 Encryption Algorithm (From RFC-1320, section 3.4)

3.4 Step 4. Process Message in 16-Word Blocks

We first define four auxiliary functions that each take as input three 32-bit words and produce as output one 32-bit word.

```
F(X,Y,Z) = XY v not(X) Z
G(X,Y,Z) = XZ v Y not(Z)
H(X,Y,Z) = X xor Y xor Z
I(X,Y,Z) = Y xor (X v not(Z))
```

In each bit position F acts as a conditional: if X then Y else Z. The function F could have been defined using + instead of v since XY and not(X)Z will never have 1's in the same bit position.) It is interesting to note that if the bits of X, Y, and Z are independent and unbiased, the each bit of F(X,Y,Z) will be independent and unbiased.

The functions G, H, and I are similar to the function F, in that they act in "bitwise parallel" to produce their output from the bits of X, Y, and Z, in such a manner that if the corresponding bits of X, Y, and Z are independent and unbiased, then each bit of G(X,Y,Z), H(X,Y,Z), and I(X,Y,Z) will be independent and unbiased. Note that the function H is the bit-wise "xor" or "parity" function of its inputs.

This step uses a 64-element table T[1 ... 64] constructed from the sine function. Let T[i] denote the i-th element of the table, which is equal to the integer part of 4294967296 times abs(sin(i)), where i is in radians. The elements of the table are given in the appendix.

Do the following:

```
/* Process each 16-word block. */
For i = 0 to N/16-1 do

  /* Copy block i into X. */
  For j = 0 to 15 do
    Set X[j] to M[i*16+j].
  end /* of loop on j */
```

```
/* Save A as AA, B as BB, C as CC, and D as DD. */
AA = A
BB = B
CC = C
DD = D

/* Round 1. */
/* Let [abcd k s i] denote the operation
     a = b + ((a + F(b,c,d) + X[k] + T[i]) <<< s). */
/* Do the following 16 operations. */
[ABCD  0  7  1]  [DABC  1 12  2]  [CDAB  2 17  3]  [BCDA  3 22  4]
[ABCD  4  7  5]  [DABC  5 12  6]  [CDAB  6 17  7]  [BCDA  7 22  8]
[ABCD  8  7  9]  [DABC  9 12 10]  [CDAB 10 17 11]  [BCDA 11 22 12]
[ABCD 12  7 13]  [DABC 13 12 14]  [CDAB 14 17 15]  [BCDA 15 22 16]

/* Round 2. */
/* Let [abcd k s i] denote the operation
     a = b + ((a + G(b,c,d) + X[k] + T[i]) <<< s). */
/* Do the following 16 operations. */
[ABCD  1  5 17]  [DABC  6  9 18]  [CDAB 11 14 19]  [BCDA  0 20 20]
[ABCD  5  5 21]  [DABC 10  9 22]  [CDAB 15 14 23]  [BCDA  4 20 24]
[ABCD  9  5 25]  [DABC 14  9 26]  [CDAB  3 14 27]  [BCDA  8 20 28]
[ABCD 13  5 29]  [DABC  2  9 30]  [CDAB  7 14 31]  [BCDA 12 20 32]

/* Round 3. */
/* Let [abcd k s t] denote the operation
     a = b + ((a + H(b,c,d) + X[k] + T[i]) <<< s). */
/* Do the following 16 operations. */
[ABCD  5  4 33]  [DABC  8 11 34]  [CDAB 11 16 35]  [BCDA 14 23 36]
[ABCD  1  4 37]  [DABC  4 11 38]  [CDAB  7 16 39]  [BCDA 10 23 40]
[ABCD 13  4 41]  [DABC  0 11 42]  [CDAB  3 16 43]  [BCDA  6 23 44]
[ABCD  9  4 45]  [DABC 12 11 46]  [CDAB 15 16 47]  [BCDA  2 23 48]

/* Round 4. */
/* Let [abcd k s t] denote the operation
     a = b + ((a + I(b,c,d) + X[k] + T[i]) <<< s). */
/* Do the following 16 operations. */
[ABCD  0  6 49]  [DABC  7 10 50]  [CDAB 14 15 51]  [BCDA  5 21 52]
[ABCD 12  6 53]  [DABC  3 10 54]  [CDAB 10 15 55]  [BCDA  1 21 56]
[ABCD  8  6 57]  [DABC 15 10 58]  [CDAB  6 15 59]  [BCDA 13 21 60]
[ABCD  4  6 61]  [DABC 11 10 62]  [CDAB  2 15 63]  [BCDA  9 21 64]
```

```
  /* Then perform the following additions. (That is increment each
     of the four registers by the value it had before this block
     was started.) */
  A = A + AA
  B = B + BB
  C = C + CC
  D = D + DD

end /* of loop on i */
```

References

[1] Cambridge Security Workshop. *Description of a New Variable-Length Key, 64-bit block cipher*, December 1994.

[2] Microsoft Corporation. Microsoft ntlm. *Microsoft Developer Network*.

[3] Microsoft Corporation. User authentication with windows NT. *Microsoft Knowledge Base*, (102716), 2001.

[4] Microsoft Corporation. Storing passwords using reversible encryption. *Microsoft Documentation Resource*, (102716), 2004.

[5] Hans Dobbertin. Cryptanalysis of md4. *Journal of Cryptology*, 11:253–271, 1998.

[6] Robert Morris and Ken Thompson. Password security: A case history.

[7] Data encryption standard (des). (FIPS 46-2), 1988.

[8] Niels Provos and David Mazières. A future-adaptable password scheme. Technical report, The OpenBSD Project.

[9] R. Rivest. Rfc 1320 - the md4 digest algorithm, 1992.

[10] R. Rivest. Rfc 1321 - the md5 digest algorithm, 1992.