                     Microsoft PPP CHAP Extensions

Status of this Memo

   This memo provides information for the Internet community. It does
   not specify an Internet standard of any kind. Distribution of this
   memo is unlimited.

IESG Note

   The protocol described here has significant vulnerabilities. People
   planning on implementing or using this protocol should read section
   12, "Security Considerations".

1. Abstract

   The Point-to-Point Protocol (PPP) [1] provides a standard method for
   transporting multi-protocol datagrams over point-to-point links. PPP
   defines an extensible Link Control Protocol and a family of Network
   Control Protocols (NCPs) for establishing and configuring different
   network-layer protocols.

   This document describes Microsoft's PPP CHAP dialect (MS-CHAP), which
   extends the user authentication functionality provided on Windows
   networks to remote workstations. MS-CHAP is closely derived from the
   PPP Challenge Handshake Authentication Protocol described in RFC 1994
   [2], which the reader should have at hand.

   The algorithms used in the generation of various MS-CHAP protocol
   fields are described in an appendix.

2. Introduction

   Microsoft created MS-CHAP to authenticate remote Windows
   workstations, providing the functionality to which LAN-based users
   are accustomed while integrating the encryption and hashing
   algorithms used on Windows networks.

Where possible, MS-CHAP is consistent with standard CHAP. Briefly,
the differences between MS-CHAP and standard CHAP are:

   * MS-CHAP is enabled by negotiating CHAP Algorithm 0x80 in LCP
     option 3, Authentication Protocol.

   * The MS-CHAP Response packet is in a format designed for
     compatibility with Microsoft's Windows NT 3.5, 3.51 and 4.0, and
     Windows95 networking products. The MS-CHAP format does not
     require the authenticator to store a clear-text or reversibly
     encrypted password.

   * MS-CHAP provides authenticator-controlled authentication retry
     and password changing mechanisms.

   * MS-CHAP defines a set of reason-for-failure codes returned in
     the Failure packet Message field.

3. Specification of Requirements

   In this document, the key words "MAY", "MUST, "MUST NOT", "optional",
   "recommended", "SHOULD", and "SHOULD NOT" are to be interpreted as
   described in [2].

4. LCP Configuration

   The LCP configuration for MS-CHAP is identical to that for standard
   CHAP, except that the Algorithm field has value 0x80, rather than the
   MD5 value 0x05. PPP implementations which do not support MS-CHAP,
   but correctly implement LCP Config-Rej, should have no problem
   dealing with this non-standard option.

5. Challenge Packet

   The MS-CHAP Challenge packet is identical in format to the standard
   CHAP Challenge packet.

   MS-CHAP authenticators send an 8-octet challenge Value field. Peers
   need not duplicate Microsoft's algorithm for selecting the 8-octet
   value, but the standard guidelines on randomness [1,2,7] SHOULD be
   observed.

   Microsoft authenticators do not currently provide information in the
   Name field. This may change in the future.

6. Response Packet

   The MS-CHAP Response packet is identical in format to the standard
   CHAP Response packet. However, the Value field is sub-formatted
   differently as follows:

      24 octets: LAN Manager compatible challenge response
      24 octets: Windows NT compatible challenge response
       1 octet : "Use Windows NT compatible challenge response" flag

   The LAN Manager compatible challenge response is an encoded function
   of the password and the received challenge as output by the routine
   LmChallengeResponse() (see section A.1, below). LAN Manager
   passwords are limited to 14 case-insensitive OEM characters. Note
   that use of the LAN Manager compatible challenge response has been
   deprecated; peers SHOULD NOT generate it, and the sub-field SHOULD be
   zero-filled. The algorithm used in the generation of the LAN Manager
   compatible challenge response is described here for informational
   purposes only.

   The Windows NT compatible challenge response is an encoded function
   of the password and the received challenge as output by the routine
   NTChallengeResponse() (see section A.5, below). The Windows NT
   password is a string of 0 to (theoretically) 256 case-sensitive
   Unicode [8] characters. Current versions of Windows NT limit
   passwords to 14 characters, mainly for compatibility reasons; this
   may change in the future.

   The "use Windows NT compatible challenge response" flag, if 1,
   indicates that the Windows NT response is provided and should be used
   in preference to the LAN Manager response. The LAN Manager response
   will still be used if the account does not have a Windows NT password
   hash, e.g. if the password has not been changed since the account
   was uploaded from a LAN Manager 2.x account database. If the flag is
   0, the Windows NT response is ignored and the LAN Manager response is
   used. Since the use of LAN Manager authentication has been
   deprecated, this flag SHOULD always be set (1) and the LAN Manager
   compatible challenge response field SHOULD be zero-filled.

   The Name field identifies the peer's user account name. The Windows
   NT domain name may prefix the user's account name (e.g.
   "BIGCO\johndoe" where "BIGCO" is a Windows NT domain containing the
   user account "john-doe"). If a domain is not provided, the backslash
   should also be omitted, (e.g. "johndoe").

7. Success Packet

   The Success packet is identical in format to the standard CHAP
   Success packet.

8. Failure Packet

   The Failure packet is identical in format to the standard CHAP
   Failure packet. There is, however, formatted text stored in the
   Message field which, contrary to the standard CHAP rules, affects the
   protocol. The Message field format is:

        "E=eeeeeeeee R=r C=cccccccccccccccc V=vvvvvvvvvv"

      where

         The "eeeeeeeee" is the decimal error code (need not be 10
         digits) corresponding to one of those listed below, though
         implementations should deal with codes not on this list
         gracefully.

            646 ERROR_RESTRICTED_LOGON_HOURS
            647 ERROR_ACCT_DISABLED
            648 ERROR_PASSWD_EXPIRED
            649 ERROR_NO_DIALIN_PERMISSION
            691 ERROR_AUTHENTICATION_FAILURE
            709 ERROR_CHANGING_PASSWORD

         The "r" is a flag set to "1" if a retry is allowed, and "0" if
         not. When the authenticator sets this flag to "1" it disables
         short timeouts, expecting the peer to prompt the user for new
         credentials and resubmit the response.

         The "cccccccccccccccc" is 16 hexadecimal digits representing an
         ASCII representation of a new challenge value. This field is
         optional. If it is not sent, the authenticator expects the
         resubmitted response to be calculated based on the previous
         challenge value plus decimal 23 in the first octet, i.e. the
         one immediately following the Value Size field. Windows 95
         authenticators may send this field. Windows NT authenticators
         do not, but may in the future. Both systems implement peer
         support of this field.

         The "vvvvvvvvvv" is the decimal version code (need not be 10
         digits) indicating the MS-CHAP protocol version supported on
         the server. Currently, this is interesting only in selecting a
         Change Password packet type. If the field is not present the
         version should be assumed to be 1; since use of the version 1

Change Password packet has been deprecated, this field SHOULD
always contain a value greater than or equal to 2.

Implementations should accept but ignore additional text they do not
recognize.

9. Change Password Packet (version 1)

The version 1 Change Password packet does not appear in standard
CHAP. It allows the peer to change the password on the account
specified in the previous Response packet. The version 1 Change
Password packet should be sent only if the authenticator reports
ERROR_PASSWD_EXPIRED (E=648) and V is either missing or equal to one
in the Message field of the Failure packet.

The use of the Change Password Packet (version 1) has been
deprecated; the format of the packet is described here for
informational purposes, but peers SHOULD NOT transmit it.

The format of this packet is as follows:

```
 1 octet : Code (=5)
 1 octet : Identifier
 2 octets: Length (=72)
16 octets: Encrypted LAN Manager Old password Hash
16 octets: Encrypted LAN Manager New Password Hash
16 octets: Encrypted Windows NT Old Password Hash
16 octets: Encrypted Windows NT New Password Hash
 2 octets: Password Length
 2 octets: Flags
```

Code
   5

Identifier
   The Identifier field is one octet and aids in matching requests
   and replies. The value is the Identifier of the received
   Failure packet to which this packet responds plus 1.

Length
   72

Encrypted LAN Manager New Password Hash
Encrypted LAN Manager Old Password Hash
   These fields contain the LAN Manager password hash of the new
   and old passwords encrypted with the last received challenge
   value, as output by the routine LmEncryptedPasswordHash() (see
   section A.8, below).

Encrypted Windows NT New Password Hash
Encrypted Windows NT Old Password Hash
    These fields contain the Windows NT password hash of the new
    and old passwords encrypted with the last received challenge
    value, as output by the pseudo-code routine
    NtEncryptedPasswordHash() (see section A.10, below).

Password Length
    The length in octets of the LAN Manager compatible form of the
    new password. If this value is greater than or equal to zero
    and less than or equal to 14 it is assumed that the encrypted
    LAN Manager password hash fields are valid. Otherwise, it is
    assumed these fields are not valid, in which case the Windows
    NT compatible passwords MUST be provided.

Flags
    This field is two octets in length. It is a bit field of
    option flags where 0 is the least significant bit of the 16-bit
    quantity:

        Bit 0
            If this bit is set (1), it indicates that the encrypted
            Windows NT hashed passwords are valid and should be used.
            If this bit is cleared (0), the Windows NT fields are not
            used and the LAN Manager fields must be provided.

        Bits 1-15
            Reserved, always clear (0).

10. Change Password Packet (version 2)

   The version 2 Change Password packet does not appear in standard
   CHAP. It allows the peer to change the password on the account
   specified in the preceding Response packet. The version 2 Change
   Password packet should be sent only if the authenticator reports
   ERROR_PASSWD_EXPIRED (E=648) and a version of 2 or greater in the
   Message field of the Failure packet.

   This packet type is supported by Windows NT 3.51, 4.0 and recent
   versions of Windows 95. It is not supported by Windows NT 3.5 or
   early versions of Windows 95.

    The format of this packet is as follows:

        1 octet : Code
        1 octet : Identifier
        2 octets : Length
      516 octets : Password Encrypted with Old NT Hash

```
         16 octets : Old NT Hash Encrypted with New NT Hash
        516 octets : Password Encrypted with Old LM Hash
         16 octets : Old LM Hash Encrypted With New NT Hash
         24 octets : LAN Manager compatible challenge response
         24 octets : Windows NT compatible challenge response
          2-octet  : Flags
```

   Code
      6

   Identifier
      The Identifier field is one octet and aids in matching requests
      and replies. The value is the Identifier of the received
      Failure packet to which this packet responds plus 1.

   Length
      1118

   Password Encrypted with Old NT Hash
      This field contains the PWBLOCK form of the new Windows NT
      password encrypted with the old Windows NT password hash, as
      output by the NewPasswordEncryptedWithOldNtPasswordHash()
      routine (see section A.11, below).

   Old NT Hash Encrypted with New NT Hash
      This field contains the old Windows NT password hash encrypted
      with the new Windows NT password hash, as output by the
      OldNtPasswordHashEncryptedWithNewNtPasswordHash() routine (see
      section A.14, below).

   Password Encrypted with Old LM Hash
      This field contains the PWBLOCK form of the new Windows NT
      password encrypted with the old LAN Manager password hash, as
      output by the NewPasswordEncryptedWithOldLmPasswordHash()
      routine described in section A.15, below. Note, however, that
      the use of this field has been deprecated: peers SHOULD NOT
      generate it, and this field SHOULD be zero-filled.

   Old LM Hash Encrypted With New NT Hash
      This field contains the old LAN Manager password hash encrypted
      with the new Windows NT password hash, as output by the
      OldLmPasswordHashEncryptedWithNewNtPasswordHash() routine (see
      section A.16, below). Note, however, that the use of this
      field has been deprecated: peers SHOULD NOT generate it, and
      this field SHOULD be zero-filled.
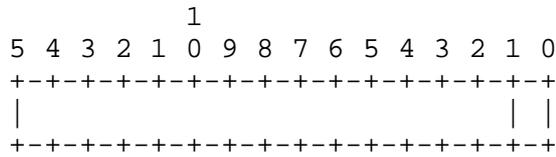
          LAN Manager compatible challenge response
          Windows NT compatible challenge response
             The challenge response field (as described in the Response
             packet description), but calculated on the new password and the
             same challenge used in the last response. Note that use of the
             LAN Manager compatible challenge response has been deprecated;
             peers SHOULD NOT generate it, and the field SHOULD be zero-
             filled.

       Flags
          This field is two octets in length. It is a bit field of
          option flags where 0 is the least significant bit of the 16-bit
          quantity. The format of this field is illustrated in the
          following diagram:

                   1
          5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0
          +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
          | | | |
          +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

             Bit 0
                The "use Windows NT compatible challenge response" flag
                as described in the Response packet.

             Bit 1
                Set (1) indicates that the "Password Encrypted with Old
                LM Hash" and "Old LM Hash Encrypted With New NT Hash"
                fields are valid and should be used. Clear (0) indicates
                these fields are not valid. This bit SHOULD always be
                clear (0).

             Bits 2-15
                Reserved, always clear (0).

11. Security Considerations

   As an implementation detail, the authenticator SHOULD limit the
   number of password retries allowed to make brute-force password
   guessing attacks more difficult.

   Because the challenge value is encrypted using the password hash to
   form the response and the challenge is transmitted in clear-text
   form, both passive known-plaintext and active chosen-plaintext
   attacks against the password hash are possible. Suitable precautions
   (i.e., frequent password changes) SHOULD be taken in environments
   where eavesdropping is likely.

The Change Password (version 1) packet is vulnerable to a passive
eavesdropping attack which can easily reveal the new password hash.
For this reason, it MUST NOT be sent if eavesdropping is possible.

12. References

   [1] Simpson, W., "The Point-to-Point Protocol (PPP)", STD 51, RFC
       1661, July 1994.

   [2] Simpson, W., "PPP Challenge Handshake Authentication Protocol
       (CHAP)", RFC 1994, August 1996.

   [3] Bradner, S., "Key words for use in RFCs to Indicate Requirement
       Levels", BCP 14, RFC 2119, March 1997.

   [4] "Data Encryption Standard (DES)", Federal Information Processing
       Standard Publication 46-2, National Institute of Standards and
       Technology, December 1993.

   [5] Rivest, R., "MD4 Message Digest Algorithm", RFC 1320, April 1992.

   [6] RC4 is a proprietary encryption algorithm available under license
       from RSA Data Security Inc. For licensing information, contact:
       RSA Data Security, Inc.
       100 Marine Parkway
       Redwood City, CA 94065-1031

   [7] Eastlake, D., Crocker, S., and J. Schiller, "Randomness
       Recomnendations for Security", RFC 1750, December 1994.

   [8] "The Unicode Standard, Version 2.0", The Unicode Consortium,
       Addison-Wesley, 1996. ISBN 0-201-48345-9.

   [9] "DES Modes of Operation", Federal Information Processing
       Standards Publication 81, National Institute of Standards and
       Technology, December 1980

13. Acknowledgements

14. Chair's Address

   The PPP Extensions Working Group can be contacted via the current
   chair:

   Karl Fox
   Ascend Communications
   3518 Riverside Drive
   Suite 101
   Columbus, OH 43221

   Phone: +1 614 326 6841
   EMail: karl@ascend.com

15. Authors' Addresses

   Questions about this memo can also be directed to:

   Glen Zorn
   Microsoft Corporation
   One Microsoft Way
   Redmond, Washington 98052

   Phone: +1 425 703 1559
   Fax: +1 425 936 7329
   EMail: glennz@microsoft.com

   Steve Cobb
   Microsoft Corporation
   One Microsoft Way
   Redmond, Washington 98052

   EMail: stevec@microsoft.com

Appendix A - Pseudocode

   The routines mentioned in the text are described in pseudocode below.

A.1 LmChallengeResponse()

```
   LmChallengeResponse(
   IN  8-octet        Challenge,
   IN  0-to-14-oem-char Password,
   OUT 24-octet       Response )
   {
      LmPasswordHash( Password, giving PasswordHash )
      ChallengeResponse( Challenge, PasswordHash, giving Response )
   }
```

A.2 LmPasswordHash()

```
   LmPasswordHash(
   IN  0-to-14-oem-char Password,
   OUT 16-octet       PasswordHash )
   {
      Set UcasePassword to the uppercased Password
      Zero pad UcasePassword to 14 characters

      DesHash( 1st 7-octets of UcasePassword,
               giving 1st 8-octets of PasswordHash )

      DesHash( 2nd 7-octets of UcasePassword,
               giving 2nd 8-octets of PasswordHash )
   }
```

A.3 DesHash()

```
   DesHash(
   IN  7-octet Clear,
   OUT 8-octet Cypher )
   {
      /*
       * Make Cypher an irreversibly encrypted form of Clear by
       * encrypting known text using Clear as the secret key.
       * The known text consists of the string
       *
       *              KGS!@#$%
       */

      Set StdText to "KGS!@#$%"
```

```
      DesEncrypt( StdText, Clear, giving Cypher )
   }
```

A.4 DesEncrypt()

```
   DesEncrypt(
   IN  8-octet Clear,
   IN  7-octet Key,
   OUT 8-octet Cypher )
   {
      /*
       * Use the DES encryption algorithm [4] in ECB mode [9]
       * to encrypt Clear into Cypher such that Cypher can
       * only be decrypted back to Clear by providing Key.
       * Note that the DES algorithm takes as input a 64-bit
       * stream where the 8th, 16th, 24th, etc.  bits are
       * parity bits ignored by the encrypting algorithm.
       * Unless you write your own DES to accept 56-bit input
       * without parity, you will need to insert the parity bits
       * yourself.
       */
   }
```

A.5 NtChallengeResponse()

```
   NtChallengeResponse(
   IN  8-octet               Challenge,
   IN  0-to-256-unicode-char Password,
   OUT 24-octet              Response )
   {
      NtPasswordHash( Password, giving PasswordHash )
      ChallengeResponse( Challenge, PasswordHash, giving Response )
   }
```

A.6 NtPasswordHash()

```
   NtPasswordHash(
   IN  0-to-256-unicode-char Password,
   OUT 16-octet              PasswordHash )
   {
      /*
       * Use the MD4 algorithm [5] to irreversibly hash Password
       * into PasswordHash.  Only the password is hashed without
       * including any terminating 0.
       */
```

```
   }


A.7 ChallengeResponse()

   ChallengeResponse(
   IN  8-octet  Challenge,
   IN  16-octet PasswordHash,
   OUT 24-octet Response )
   {
      Set ZPasswordHash to PasswordHash zero-padded to 21 octets

      DesEncrypt( Challenge,
                  1st 7-octets of ZPasswordHash,
                  giving 1st 8-octets of Response )

      DesEncrypt( Challenge,
                  2nd 7-octets of ZPasswordHash,
                  giving 2nd 8-octets of Response )

      DesEncrypt( Challenge,
                  3rd 7-octets of ZPasswordHash,
                  giving 3rd 8-octets of Response )
   }


A.8 LmEncryptedPasswordHash()

   LmEncryptedPasswordHash(
   IN  0-to-14-oem-char Password,
   IN  8-octet          KeyValue,
   OUT 16-octet         Cypher )
   {
      LmPasswordHash( Password, giving PasswordHash )

      PasswordHashEncryptedWithBlock( PasswordHash,
                                      KeyValue,
                                      giving Cypher )
   }


A.9 PasswordHashEncryptedWithBlock()

   PasswordHashEncryptedWithBlock(
   IN  16-octet PasswordHash,
   IN  8-octet  Block,
   OUT 16-octet Cypher )
   {
```

```
      DesEncrypt( 1st 8-octets PasswordHash,
                  1st 7-octets Block,
                  giving 1st 8-octets Cypher )

      DesEncrypt( 2nd 8-octets PasswordHash,
                  1st 7-octets Block,
                  giving 2nd 8-octets Cypher )
   }
```


A.10 NtEncryptedPasswordHash()

```
   NtEncryptedPasswordHash( IN   0-to-14-oem-char  Password IN  8-octet
   Challenge OUT 16-octet         Cypher ) {
      NtPasswordHash( Password, giving PasswordHash )

      PasswordHashEncryptedWithBlock( PasswordHash,
                                      Challenge,
                                      giving Cypher )
   }
```


A.11 NewPasswordEncryptedWithOldNtPasswordHash()

```
   datatype-PWBLOCK
   {
      256-unicode-char Password
      4-octets         PasswordLength
   }

   NewPasswordEncryptedWithOldNtPasswordHash(
   IN   0-to-256-unicode-char NewPassword,
   IN   0-to-256-unicode-char OldPassword,
   OUT datatype-PWBLOCK       EncryptedPwBlock )
   {
      NtPasswordHash( OldPassword, giving PasswordHash )

      EncryptPwBlockWithPasswordHash( NewPassword,
                                      PasswordHash,
                                      giving EncryptedPwBlock )
   }
```


A.12 EncryptPwBlockWithPasswordHash()

```
   EncryptPwBlockWithPasswordHash(
   IN   0-to-256-unicode-char Password,
   IN   16-octet              PasswordHash,
```

```
   OUT datatype-PWBLOCK      PwBlock )
   {

      Fill ClearPwBlock with random octet values
      PwSize = lstrlenW( Password ) * sizeof( unicode-char )
      PwOffset = sizeof( ClearPwBlock.Password ) - PwSize
      Move PwSize octets to (ClearPwBlock.Password + PwOffset ) from Passwor
d
      ClearPwBlock.PasswordLength = PwSize
      Rc4Encrypt( ClearPwBlock,
                  sizeof( ClearPwBlock ),
                  PasswordHash,
                  sizeof( PasswordHash ),
                  giving PwBlock )
   }
```

A.13 Rc4Encrypt()

```
   Rc4Encrypt(
   IN  x-octet Clear,
   IN  integer ClearLength,
   IN  y-octet Key,
   IN  integer KeyLength,
   OUT x-octet Cypher )
   {
      /*
       * Use the RC4 encryption algorithm [6] to encrypt Clear of
       * length ClearLength octets into a Cypher of the same length
       * such that the Cypher can only be decrypted back to Clear
       * by providing a Key of length KeyLength octets.
       */
   }
```

A.14 OldNtPasswordHashEncryptedWithNewNtPasswordHash()

```
   OldNtPasswordHashEncryptedWithNewNtPasswordHash(
   IN  0-to-256-unicode-char NewPassword,
   IN  0-to-256-unicode-char OldPassword,
   OUT 16-octet              EncryptedPasswordHash )
   {
      NtPasswordHash( OldPassword, giving OldPasswordHash )
      NtPasswordHash( NewPassword, giving NewPasswordHash )
      NtPasswordHashEncryptedWithBlock( OldPasswordHash,
                                        NewPasswordHash,
                                        giving EncryptedPasswordHash )
   }
```

```
A.15 NewPasswordEncryptedWithOldLmPasswordHash()

   NewPasswordEncryptedWithOldLmPasswordHash(
   IN  0-to-256-unicode-char NewPassword,
   IN  0-to-256-unicode-char OldPassword,
   OUT datatype-PWBLOCK      EncryptedPwBlock )
   {
      LmPasswordHash( OldPassword, giving PasswordHash )

      EncryptPwBlockWithPasswordHash( NewPassword, PasswordHash,
                                      giving EncryptedPwBlock )
   }


A.16 OldLmPasswordHashEncryptedWithNewNtPasswordHash()

   OldLmPasswordHashEncryptedWithNewNtPasswordHash(
   IN  0-to-256-unicode-char NewPassword,
   IN  0-to-256-unicode-char OldPassword,
   OUT 16-octet             EncryptedPasswordHash )
   {
      LmPasswordHash( OldPassword, giving OldPasswordHash )

      NtPasswordHash( NewPassword, giving NewPasswordHash )

      NtPasswordHashEncryptedWithBlock( OldPasswordHash, NewPasswordHash,
                                      giving EncrytptedPasswordHash )
   }


A.17 NtPasswordHashEncryptedWithBlock()

   NtPasswordHashEncryptedWithBlock(
   IN  16-octet PasswordHash,
   IN  16-octet Block,
   OUT 16-octet Cypher )
   {
      DesEncrypt( 1st 8-octets PasswordHash,
                  1st 7-octets Block,
                  giving 1st 8-octets Cypher )

      DesEncrypt( 2nd 8-octets PasswordHash,
                  2nd 7-octets Block,
                  giving 2nd 8-octets Cypher )
   }
```

Appendix B - Examples

B.1 Negotiation Examples

   Here are some examples of typical negotiations. The peer is on the
   left and the authenticator is on the right.

   The packet sequence ID is incremented on each authentication retry
   Response and on the change password response. All cases where the
   packet sequence ID is updated are noted below.

   Response retry is never allowed after Change Password. Change
   Password may occur after Response retry. The implied challenge form
   is shown in the examples, though all cases of "first challenge+23"
   should be replaced by the "C=cccccccccccccccc" challenge if
   authenticator supplies it in the Failure packet.

B.1.1 Successful authentication

           <- Challenge
        Response ->
           <- Success

B.1.2 Failed authentication with no retry allowed

           <- Challenge
        Response ->
           <- Failure (E=691 R=0)

B.1.3 Successful authentication after retry

           <- Challenge
        Response ->
           <- Failure (E=691 R=1), disable short timeout
        Response (++ID) to first challenge+23 ->
           <- Success

B.1.4 Failed hack attack with 3 attempts allowed

           <- Challenge
        Response ->
           <- Failure (E=691 R=1), disable short timeout
        Response (++ID) to first challenge+23 ->
           <- Failure (E=691 R=1), disable short timeout
        Response (++ID) to first challenge+23+23 ->

```
                <- Failure (E=691 R=0)
```

B.1.5 Successful authentication with password change

```
            <- Challenge
        Response ->
            <- Failure (E=648 R=0 V=2), disable short timeout
        ChangePassword (++ID) to first challenge ->
            <- Success
```

B.1.6 Successful authentication with retry and password change

```
            <- Challenge
        Response ->
            <- Failure (E=691 R=1), disable short timeout
        Response (++ID) to first challenge+23 ->
            <- Failure (E=648 R=0 V=2), disable short timeout
        ChangePassword (++ID) to first challenge+23 ->
            <- Success
```

B.2 Hash Example

Intermediate values for password "MyPw".

```
   8-octet Challenge:
   10 2D B5 DF 08 5D 30 41

   0-to-256-unicode-char NtPassword:
   4D 00 79 00 50 00 77 00

   16-octet NtPasswordHash:
   FC 15 6A F7 ED CD 6C 0E DD E3 33 7D 42 7F 4E AC

   24-octet NtChallengeResponse:
   4E 9D 3C 8F 9C FD 38 5D 5B F4 D3 24 67 91 95 6C
   A4 C3 51 AB 40 9A 3D 61
```

B.3 Example of DES Key Generation

DES uses 56-bit keys, expanded to 64 bits by the insertion of parity
bits. After the parity of the key has been fixed, every eighth bit is a
parity bit and the number of bits that are set (1) in each octet is odd;
i.e., odd parity. Note that many DES engines do not check parity,
however, simply stripping the parity bits. The following example
illustrates the values resulting from the use of the 16-octet
NTPasswordHash shown in Appendix B.2 to generate a pair of DES keys

(e.g., for use in the NtPasswordHashEncryptedWithBlock() described in
Appendix A.17).

    16-octet NtPasswordHash:
    FC 15 6A F7 ED CD 6C 0E DD E3 33 7D 42 7F 4E AC

    First "raw" DES key (initial 7 octets of password hash):
    FC 15 6A F7 ED CD 6C

    First parity-corrected DES key (eight octets):
    FD 0B 5B 5E 7F 6E 34 D9

    Second "raw" DES key (second 7 octets of password hash)
    0E DD E3 33 7D 42 7F

    Second parity-corrected DES key (eight octets):
    0E 6E 79 67 37 EA 08 FE

Full Copyright Statement

   Copyright (C) The Internet Society (1998). All Rights Reserved.

   This document and translations of it may be copied and furnished to
   others, and derivative works that comment on or otherwise explain it
   or assist in its implementation may be prepared, copied, published
   and distributed, in whole or in part, without restriction of any
   kind, provided that the above copyright notice and this paragraph are
   included on all such copies and derivative works. However, this
   document itself may not be modified in any way, such as by removing
   the copyright notice or references to the Internet Society or other
   Internet organizations, except as needed for the purpose of
   developing Internet standards in which case the procedures for
   copyrights defined in the Internet Standards process must be
   followed, or as required to translate it into languages other than
   English.

   The limited permissions granted above are perpetual and will not be
   revoked by the Internet Society or its successors or assigns.

   This document and the information contained herein is provided on an
   "AS IS" basis and THE INTERNET SOCIETY AND THE INTERNET ENGINEERING
   TASK FORCE DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING
   BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION
   HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF
   MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.