IceWall SSO

Version 10.0

Password Encryption Library Developer's Manual

August 2010

Printed in Japan

HP Part No. B1544-97012

Rev.111012A

Notice

The information contained in this document is subject to change without notice.

Meticulous care has been taken in the preparation of this document, however, if a questionable or erroneous item, or an omission of content is found, please contact us.

Hewlett-Packard shall not be liable for errors contained herein or for incidental or consequential damage in connection with the furnishing, performance or use of this document.

This manual and media, such as the CD-ROM supplied as a part of the product's package, are only to be used with this product.

– Table of Contents –

1 Introduction 1
2 What is Password Encryption Library? 2
3 Development Procedure 3
3.1 Determining Encryption and Decryption Algorithm 3
3.2 Coding and Building 3
3.3 Testing and Debugging 3
4 Library Specifications 5
IW_DB_Hash 6
IW_DB_PassCheck 7
5 Remarks 8
5.1 Scope of Encryption Library 8
5.2 Performance Degrading 8
5.3 Changing Library during System Operation 8
6 Restrictions 9
7 Reference 10
7.1 Skeleton source (iwphash.c) 10
7.2 Header file (iwphash.h) 10
7.3 Makefile (64-bit HP-UX Itanium: Makefile) 11
7.4 Makefile (64-bit Linux: Makefile) 11

1 Introduction

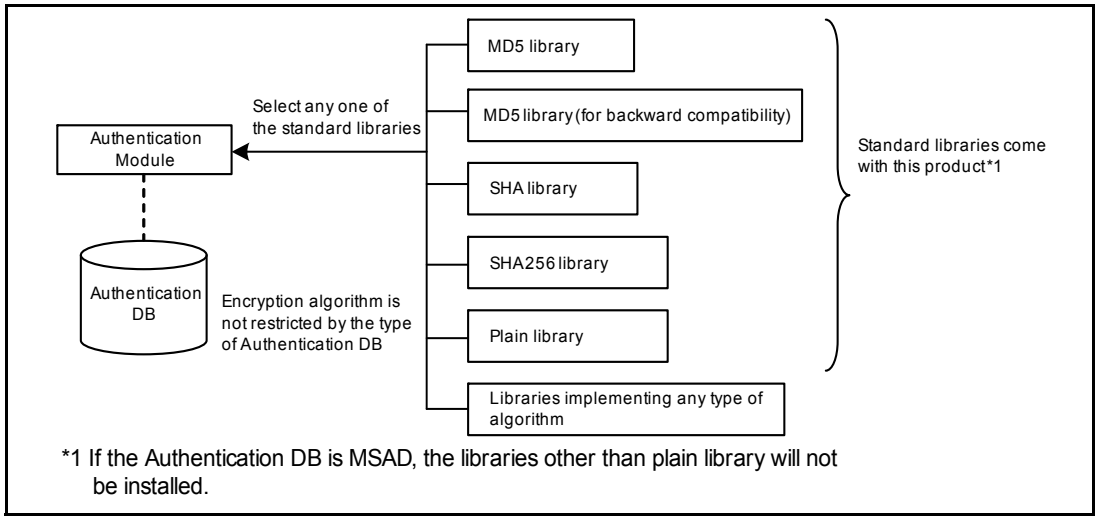
This manual provides information required to develop a library for encrypting user passwords saved in the authentication database.

Note that this manual refers to the term “encryption” as “irreversible hash algorithm.”

Notice
If you use Microsoft Active Directory as the authentication database, you cannot change the password encryption library.

2 What is Password Encryption Library?

IceWall SSO allows for selecting a password encryption algorithm by specifying it in the configuration file.



The password encryption library is implemented as a shared library. If you want to use an encryption algorithm other than those of the standard encryption library, you can also develop your own shared library with the algorithm you need.

3 Development Procedure

This section describes the general procedure for developing a password encryption library.

3.1 Determining Encryption and Decryption Algorithm

Determine the algorithm for encrypting/decrypting passwords.

3.2 Coding and Building

According to the determined algorithm, write the source code of the password encryption library, and then build it into the shared library.

Use the skeleton source code and makefile included in the development kit, for coding and building, respectively.

```
$ cd /opt/icewall-sso/developkit/certd/PwdLib
$ make
```

3.3 Testing and Debugging

Test and debug the developed shared library using a test program and other tools. After the shared library has passed the test, integrate it into the Authentication Module and then perform an operation test.

Integrate the shared library into the Authentication Module in the following procedure.

(1) Stop the Authentication Module.

```
$ /opt/icewall-sso/certd/bin/end-cert
```

(2) Delete the link to the standard password encryption library installed with the product by default.

```
$ rm /opt/icewall-sso/lib/certd/libPH.sl
```

(3) Define the link to the developed password encryption library.

HP-UX 11i v3 (Itanium) and Linux

```
$ ln -s /opt/icewall-sso/developkit/certd/PwdLib/libiwPH.sl /
opt/icewall-sso/lib/certd/libPH.so
```

(4) Start the Authentication Module.

```
$ /opt/icewall-sso/certd/bin/start-cert
```

4 Library Specifications

For the password encryption library, you need to develop two interface functions according to the specifications mentioned later.

- Password Encryption Function (IW_DB_Hash)
- Encrypted Password Comparison Function (IW_DB_PassCheck)

The specifications of the password encryption function are described below.

(1) Encrypted password data must be preceded by the algorithm identification code enclosed in “{}.”
(Example) {MyHash}Encrypted password data

(2) Encrypted password data must be 128 bytes or less in length (including the algorithm identification code).

IW_DB_Hash

This interface function encrypts a password entered by the user when logging into the system or changing the current password.

Format

int IW_DB_Hash(char* passwd, char* hashdata)

Argument

passwd : A pointer to the buffer where the password entered by the user is stored. The size of this buffer can be 1 to 128 bytes.

hashdata : A pointer to the buffer where the encrypted password is stored. The size of this buffer is fixed to 128 bytes.

Return value

The function returns one of the following values.
PH_OK: Normal termination
PH_NG: Abnormal termination

Description

The following describes the basic processing flow of this interface function.
(1) Obtain the password from the passwd argument.
(2) Encrypt the obtained password.
(3) Store the encrypted password to the hashdata argument.
(Add a NULL character '¥0' to the end of the encrypted password.)

The buffer specified as an argument to the interface function is initialized by the Authentication Module before being passed to the function. Since the maximum length of each argument (passwd and hashdata) is 129 bytes (128 bytes + NULL), be very careful when handling this area.
The operation is not guaranteed if data larger than the maximum buffer size is stored.

IW_DB_PassCheck

This interface function compares the encrypted password entered by the user, when logging into the system or changing the password, with the encrypted password stored in the authentication database.

Format

int IW_DB_PassCheck(char* inpass, char* dbpass)

Argument

inpass : A pointer to the buffer where the encrypted version of the password entered by the user is stored.

dbpass : A pointer to the buffer where the encrypted version of the password stored in the authentication database is stored.

Return value

The function returns one of the following values.
PH_OK: The entered password is the same as the saved password.
PH_NG: The entered password is different than the saved password.

Description

The basic processing flow of this interface function is described below.
(1) Compare the password given in the inpass argument with the password given in the dbpass argument.
(2) Return the comparison result.

The maximum size of the interface function arguments, inpass and bdpass, is 129 bytes and cannot be changed. (They are read-only buffers and the operation is not guaranteed if they are modified.)

5 Remarks

Note the following items when developing the password encryption library:

5.1 Scope of Encryption Library

Since the process running in the encryption library is closely linked to the internal processing of the Authentication Module, the entire process may abort when a system error occurs within the library. Be sure to thoroughly test the developed library before integrating it into the Authentication Module.

Also, do not describe any process that affects the internal processing of the Authentication Module.

5.2 Performance Degrading

Adding a new step to process to the encryption library decreases the performance of the entire process since the new step introduces an additional load to the normal operation. Therefore, to the extent possible it is recommended that complex processing be avoided within the encryption library.

5.3 Changing Library during System Operation

If the encryption library being used is changed in the middle of system operation, the users that were registered in the authentication database before the library was changed will not be able to log into the system because their passwords will not match the saved encrypted passwords.

6 Restrictions

Note the following restrictions when developing the password encryption library: Otherwise, the entire process may become unable to run or become unstable.

(1) Link a 64-bit library with the Authentication Module, which is also 64-bit.

(2) Use the thread-safe versions of the standard library functions and user functions. The Authentication Module uses threads for internal processing. The encryption library is called from the Authentication Module as a function within a thread. This is because you need to use the thread-safe versions of the system calls within the library.

(3) Do not link different versions of the standard libraries within an archive. The Authentication Module is designed to operate using libraries dynamically. However, if an archive library is linked to the encryption library, the library linked with the encryption library is used and a mismatch due to different library versions may occur. (Trouble that cannot be resolved by applying patches to the OS may occur.)

Note: In particular, you should be very careful about the state of the linked library when using a library for which configuration (i.e., configuration of libclntsh.sl) differs depending on the installation environment (e.g., Oracle) used within the encryption library.

7 Reference

The following shows the contents of the standard password encryption library development kit installed with the product by default.
The development kit consists of the following directories and files.

Directories and files					Description
/opt/icewall-sso	/developkit	/certd	/PwdLib	/iwphash.c	Password encryption library development kit
				/iwphash.h	
				/Makefile	

7.1 Skeleton source (iwphash.c)

```
#include <stdio.h>
#include "iwphash.h"

int  IW_DB_Hash( passwd, hashdata )
char *passwd;
char *hashdata;
{

    strcpy( hashdata, passwd );
    return( PH_OK );

}

int    IW_DB_PassCheck( inpass, dbpass )
char   *inpass;
char   *dbpass;
{

    if( ( strcmp( inpass, dbpass ) ) != 0  ) {
        return( PH_NG );
    } else {
        return( PH_OK );
    }
}
```

7.2 Header file (iwphash.h)

```
/*---------------------------------------------------------------------*/
/*  Password Hash Interface                                            */
/*---------------------------------------------------------------------*/
#ifndef PHASH_H
#define PHASH_H

/*---------------------------------------------------------------------*/
/*  Define                                                             */
```

```
/*--------------------------------------------------------------------------*/
#define PH_OK        0          /* Success                    */
#define PH_NG       -1          /* Failure                    */

#endif /* #ifndef PHASH_H */
```

7.3 Makefile (64-bit HP-UX Itanium: Makefile)

```
CC              =   cc

CCFLAGS         =   -D_UNIX -D_HPUX_SOURCE -D_POSIX_C_SOURCE=199
506L -Aa +e -D_FILE_OFFSET_BITS=64 -z +DD64 +z

LIBS            =   -I./

MAKEFILE        =   Makefile

OBJS            =   iwphash.o

PROGRAM         =   iwPH

SRCS            =   iwphash.c

all:                $(PROGRAM)

$(PROGRAM):         $(OBJS)

                    ld -b -z -o lib$(PROGRAM).sl $(OBJS)
.c.o:
                    $(CC) $(CCFLAGS) $(LIBS) -c $(SRCS)

clean:
                    rm -f $(OBJS) lib$(PROGRAM).sl core
```

7.4 Makefile (64-bit Linux: Makefile)

```
CCa             =   gcc

CCFLAGS         =   -m64 -fPIC -DLinux -D_FILE_OFFSET_BITS=64 -
D_LARGEFILE_SOURCE -D_GNU_SOURCE

LIBS            =   -I./

MAKEFILE        =   Makefile
```

```
OBJS            =   iwphash.o

PROGRAM         =   iwPH

SRCS            =   iwphash.c

all:                $(PROGRAM)

$(PROGRAM):         $(OBJS)
                    gcc -shared -o lib$(PROGRAM).sl $(OBJS)
.c.o:
                    $(CC) $(CCFLAGS) $(LIBS) -c $(SRCS)

clean:
                    rm -f $(OBJS) lib$(PROGRAM).sl core
```