Custom password algorithm on OpenVMS using SHA256

Author: Prasad SG
Co-Author: Rupesh Shantamurty

OpenVMS Technical Journal V18

Table of contents
Overview..................................................................................................................................... 2
How are passwords generated and validated? ................................................................. 3
How to implement custom algorithm?............................................................................. 4
Replacing SYS$HASH_PASSWORD() system service...................................................... 5
Modify user algorithm in SYSUAF.DAT........................................................................ 5
Code implementation...................................................................................................... 6
Sample build/run .......................................................................................................... 15
References.................................................................................................................... 20
Appendix....................................................................................................................... 21
Overview

It is a known fact that user passwords on OpenVMS is stored using the Purdy algorithm. Purdy is a one-way hashing algorithm, which takes the user name, password, and a salt value to generate the hash.

The Purdy polynomial creates a 64-bit output hash from a set of input provided and generates a large change in the output for small changes in input. Details on Purdy algorithm can be found in the OpenVMS Journal V3 under article—“Best of Ask the Wizard”, topic, “OpenVMS System and Password Security”.

There are cases when user might want to use different algorithm for password hashing, either for having a custom security or for compliance reasons.

This article details a way in which user can override this Purdy algorithm for all or specific set of users on OpenVMS. The article explains how custom hashing algorithm can be incorporated and integrated into the current validation on OpenVMS.

An illustration with code is provided, in which SHA256 hashing algorithm is replaced for specific user on OpenVMS.
How are passwords generated and validated?

Figure 1 explains how users use different tools related to login and change password. New users are created or existing user information modified using the authorize utility.

Users generally login to the system using Telnet, SSH, or DECNET login.

Users change their password using the “set password” command or their password expires and they are prompted for new password at login prompt.

The password hash is generated when a new user is created or the existing user modifies the password. The hash thus generated is stored in the SYSUAF.DAT file along with the user record information. Every time the user tries to login, the password provided at the prompt is hashed and this hash is compared against the stored hash in SYSUAF.DAT, to validate the user.

The different tools and validation mechanism on OpenVMS mostly use the SYS$HASH_PASSWORD() system service to generate the password hash.

The SYS$HASH_PASSWORD() system service takes the below arguments:

- **Password**: The user password as a string descriptor
- **Algorithm**: The algorithm to be used for hashing

  This is an unsigned character argument, which can take values ranging from 0 to 255. The range from 0 to 127 is used by HP internally to have different algorithms, which are:

  0: UAI$K_AD_II, which is 32-bit cyclic redundancy check (CRC) used in releases prior to Virtual Address Extension (VAX) VMS Version 2.0
  1: UAI$C_Purdy, Purdy algorithm used during VAX VMS Version 2.0 field test
  2: UAI$C_Purdy_V, Purdy algorithm used in releases prior to VMS Version 5.4
  3: UAI$C_Purdy_S, Purdy algorithm used in VMS Version 5.4 and later
  4 to 127: Reserved (UAI$C_PREFERRED_ALGORITHM is defined to 127 and if sys$hash_password() receives 127 as argument, it currently uses UAI$C_Purdy_S)
  128 to 255: a unique value in this range identifies the hash algorithm, which can be site specific.

- **Salt**: A two-byte value which is unique for user and used in hashing.
- **Username**: The user name as a string descriptor.

  The algorithm argument for the SYS$HASH_PASSWORD() comes in handy to implement custom password algorithm.
How to implement custom algorithm?

To implement custom algorithm, we need to do two actions.

1. Replace the SYS$HASH_PASS$WORD() with a new SYS$HASH_PASS$WORD() system service.
   Currently, the SYS$HASH_PASS$WORD() system service does not understand algorithm argument values in the 128 to 255 range.
   A replaced SYS$HASH_PASS$WORD() can be written which calls new algorithm, based on the value of algorithm argument.

2. Modify the user record in SYSUAF.DAT to indicate that the password algorithm to be used is a new algorithm for this user.

Figure 2. Replaces SYS$HASH_PASS$WORD() system service
Replacing SYS$HASH_PASSWORD() system service

To replace the SYS$HASH_PASSWORD(), we write an execlet, which exposes SYS$HASH_PASSWORD() service and has the definition of new SYS$HASH_PASSWORD(). We load this execlet using the standard execlet loading mechanism on OpenVMS.

Once loaded, the new SYS$HASH_PASSWORD() starts getting called, instead of the original SYS$HASH_PASSWORD() provided with the operating system.

Assuming our execlet name is SYS$HASH_PASSWORD.EXE, we need to do the following:

- Copy the SYS$HASH_PASSWORD.EXE to SYS$COMMON:[SYS$LDR]
- Mark the execlet for loading. The command is:

```
$ MC SYSMAN
SYSMAN> SYS_LOADABLE ADD _LOCAL_ -
SYS$HASH_PASSWORD -
/LOAD_STEP = SYSINIT -
/SEVERITY = WARNING -
/MESSAGE = "Failure to load SYS$HASH_PASSWORD.EXE"
SYSMAN> EXIT
```

- Update the system image data file (VMS$SYSTEM_IMAGES.DATA) by the command:

```
$ @SYS$UPDATE:VMS$SYSTEM_IMAGES.COM
```

- Reboot the system. During reboot, if there is any problem in loading the execlet, you see the error message: “Failure to load SYS$HASH_PASSWORD.EXE” on the console.

Modify user algorithm in SYSUAF.DAT

Once the execlet is loaded, the next step is to mark user records in SYSUAF.DAT to be validated against the hash generated by new algorithm.

By default, the user hash is generated and validated against the Purdy algorithm. To change user password hash algorithm say for user “prasadsg”, do the following from the authorize utility:

```
$ SET DEFAULT SYS$SYSTEM:
$ RUN AUTHORIZE.EXE
UAF> MODIFY -
_UAF> PRASADSG/ALGORITHM=CURRENT=CUSTOMER=128
%UAF-I-MDFYMSG, user record(s) updated
%UAF-W-ALGNEWPWD, password(s) must be reset due to algorithm changeMark the execlet for loading. The command is
UAF> MODIFY PRASADSG/PASSWORD=samplepassword
```

Note:
The new SYS$HASH_PASSWORD() should implement new hash algorithm for algorithm number 128. The new SYS$HASH_PASSWORD() should also call OpenVMS native hash algorithm lgi$hash_password() for the standard algorithm numbers 1 to 127.

The user password also needs to be reset after changing the algorithm, so that a new password hash is generated based on the latest algorithm and stored in SYSUAF.DAT.
Code implementation

Now we know that we have to create a new system service for having a custom algorithm, how do we write this system service and what should be the logic in the new system service?

The following is a sample code and explanation, which gives you an understanding on how we achieve this:

1. HASH_PASSWORD.C: This is the execlet code which replaces the sys$hash_password() system service. The overall code flow in the replaced sys$hash_password() system service is as follows:
   a. Establish condition handler so that any Access Violation (ACCVIO) will be returned and not signaled in user mode
   b. If password is NULL, return a zero hash
   c. Probe for different arguments passed, so that the memory is accessible
   d. Check if the algorithm argument is pointing towards the Standard OpenVMS algorithm (i.e. algorithm <= 127[or UAIC$C_PREFERRED_ALGORITHM]). If yes, call the standard OpenVMS hashing algorithm implemented in the operating system through a call to _lgi$hash_password().
   e. Check if algorithm argument is 128, if yes, call SHA256 hashing through a call to cust$sha256hash_password(). cust$sha256hash_password() is implemented locally in HASH_PASSWORD.C and uses the SHA256 hashing from SHA2.C file
   f. Check if algorithm argument is 129, if yes, call cust$dummyhash_password(). This is a dummy hashing just to check how the code works

   Note:
The 128 algorithm number for SHA256 and 129 for DUMMY is an example provided here. We can use any number in the 128 to 255 range.

2. SHA2.C and SHA2.H: This is the code which does SHA256 hashing. This code was taken from the FREEBDS code tree and modified to build on OpenVMS. The entire code of SHA2.C is not provided, as this can span multiple pages in this article. Instead, the difference of code required on OpenVMS is provided.

3. HASH_PASSWORD_BLD.COM: The routines which build and link all files into an execlet. The final execlet image obtained is SYS$HASH_PASSWORD.EXE.

4. HASH_PASSWORD.OPT: The options file used during linking

Steps to build the execlet and test:
1. Place all the files in a separate directory on OpenVMS system
2. Ensure that you have HP C compiler installed
3. Run $ @HASH_PASSWORD_BLD.COM command to build
   – “SYS$HASH_PASSWORD.EXE is created in the same directory."
4. The SYS$HASH_PASSWORD.EXE execlet should get created in the same directory
5. Load this execlet using the SYSMAN utility as mentioned in “Replacing SYS$HASH_PASSWORD() system service” section.
6. Modify a test user record using the steps mentioned in “Modify user algorithm in SYSUAF.DAT” section

This code has been built and a preliminary testing done on OpenVMS Integrity V8.4 system.

Note:
Since this is an execlet code there are chances of system not rebooting back properly or logins not working. Try this out on a test system.
HASH_PASSWORD.C

    HASH_PASSWORD.C:
    **
    ** Abstract:
    **
    ** This module contains an alternate password hash algorithms
    ** implementation which uses SHA256 for OpenVMS system.
    **
    ** The current preferred algorithm on OpenVMS is Purdy and many of the
    ** components in the operating system uses the sys$hash_password()
    ** to obtain the hashing results.
    **
    ** This code provides an alternate sys$hash_password() system service
    ** which can call the standard OpenVMS hashing (Purdy) or the SHA256
    ** algorithm. An additional dummy hashing function is also provided
    ** for reference.
    **
    ** The current implementation is based on similar concept explained
    ** in OpenVMS Programming concept manual - Implementing Site-Specific
    ** Security Policies section.
    **
    ** Author: Prasad SG (prasad.sg@hp.com)
    **
    ** Revision 1.0    Prasad SG       04-Mar-2012
    **     - First revision
    **
    */

#define __NEW_STARLET       1

#include <string.h>
#include <starlet.h>
#include <ssdef.h>
#include <descrip.h>
#include <dyndef.h>
#include <prvdef.h>
#include <prdef.h>
#include <psbdef.h>
#include <pxbdef.h>
#include <tlvdef.h>
#include <issdef.h>
#include <vms_macros.h>
#include <uaidef.h>
#include "sha2.h"

extern unsigned long lgi$hash_password();
extern unsigned int EXE$SIGTORET();

typedef struct
{
    unsigned short length;
    unsigned char  dtype;
    unsigned char  class;
    unsigned char  *text;
} DESCRIPTOR;

/* Initialization function for execlet. Does nothing */
int hash_password_ext_init()
{
    return SS$_NORMAL;
}

/* Function probes the password hash buffer if a write is
 * possible here */
static int probe_hash_write (struct _generic_64 *hashValue)
{
    register int ps;

    /* Get processor status */
    ps = __PAL_RD_PS();

    /* probe hash argument args for write */
    if ((__PAL_PROBEW (hashValue, 8, ps & PR$M_PS_PRVMOD) == 0)
    {
        return SS$_ACCVIO;
    }
    return SS$_NORMAL;
}

/* Function Probes the password and username descriptors if read is
 * possible */
static int probe_read_args(DESCRIPTOR *pwd_desc, DESCRIPTOR *usrnam_desc)
{
    register int ps;

    /* Get processor status */
    ps = __PAL_RD_PS();

    /* probe different args for read */
    if ((__PAL_PROBER (pwd_desc, sizeof (DESCRIPTOR*),
                  ps & PR$M_PS_PRVMOD) == 0) ||
        (__PAL_PROBER (usrnam_desc, sizeof (DESCRIPTOR*),
                        ps & PR$M_PS_PRVMOD) == 0))
    {
        return SS$_ACCVIO;
    }

    /*ifndef __HASH_TRACE
    tr_print (("Pwd text addr: %x, Length:%d",
               pwd_desc->text, pwd_desc->length));
    tr_print ("usrnam text addr: %x, Length:%d",
               usrnam_desc->text, usrnam_desc->length));
    #endif
    if ((__PAL_PROBER (pwd_desc->text, pwd_desc->length,
                        ps & PR$M_PS_PRVMOD) == 0) ||
        (__PAL_PROBER (usrnam_desc->text, usrnam_desc->length,
                        ps & PR$M_PS_PRVMOD) == 0))
{ return SS$_ACCVIO; }
return SS$_NORMAL;
}

/*
 * Function:
 *   cust$sha256hash_password
 *
 * Abstract:
 *   Function uses the sha256 algorithm to get the hash value.
 *
 * Input Argument:
 *   pwd_desc    - The password as a descriptor
 *   salt        - The salt
 *   usernam_desc - The user name as a descriptor
 *
 * Output Argument:
 *   hashValue   - The location where hash is stored
 *
 * Returns:
 *   SS$_NORMAL
 *
 * The password, salt and user name is taken for computing the
 * SHA256 hash. We get 32 byte hash from the SHA256 algorithm.
 *
 * The SHA256 final hash result is 32 bytes long and needs to be
 * fit into 8 byte hashValue. We do this by EXOR of 8 bytes each
 * from this 32 bytes to get the final hash.
 *
 */
static int cust$sha256hash_password (DESCRIPTOR *pwd_desc,
                                  unsigned short int salt,
                                  DESCRIPTOR *usrnam_desc,
                                  unsigned char hashValue[8])
{
    SHA256_CTX ctx;
    u_int8_t results[SHA256_DIGEST_LENGTH];
    int sha256ResultIndex;
    int hashIndex;
    SHA256_Init(&ctx);
    SHA256_Update(&ctx, (u_int8_t *)pwd_desc->text, pwd_desc->length);
    SHA256_Update(&ctx, (u_int8_t *)(&salt), sizeof(unsigned short int));
    SHA256_Update(&ctx, (u_int8_t *)usrnam_desc->text, usrnam_desc->length);
    SHA256_Final(results, &ctx);

    /* EXOR the 32 bytes result into 8 bytes hashValue */
    for (sha256ResultIndex = 0, hashIndex=0;
        sha256ResultIndex < SHA256_DIGEST_LENGTH;
        sha256ResultIndex++, hashIndex++)
    {
        if (8 == hashIndex)
            hashIndex = 0;
        hashValue[hashIndex] ^= results[sha256ResultIndex];
    }
    return SS$_NORMAL;
}

/*
 * Function:
 *   cust$dummyhash_password
 *
 * Abstract:
 *   A dummy hashing algorithm.
 */
* Input Argument:
  * pwd_desc - The password as a descriptor
  * salt - The salt
  * usnam_desc - The user name as a descriptor
* Output Argument:
  * hashValue - The location where hash is stored
* Returns:
  * SS$NORMAL

* This is a dummy password hash implementation.
* It EXOR's the first 8 bytes of password and username if present
* with salt and stores it into the hashValue.
*
*/
static int cust$dummyhash_password (DESCRIPTOR *pwd_desc,
                                      unsigned short int salt,
                                      DESCRIPTOR *usrnam_desc,
                                      unsigned char hashValue[8])
{
  int hashIndex;

  for(hashIndex=0;
      hashIndex < 8 &&
      hashIndex < usnam_desc->length &&
      hashIndex < pwd_desc->length;
      hashIndex++)
  {
    hashValue[hashIndex] = usnam_desc->text[hashIndex] ^
                           pwd_desc->text[hashIndex] ^
                           salt;
  }
  return SS$NORMAL;
}

/*
 * Function:
 *    sys$hash_password
 * Abstract:
 *    Function implements an alternate sys$hash_password system service.
 *    The function takes different actions based on the “alg” (Algorithm)
 *    argument.
 *
 *    If “alg” is less than equal to UAF$C_PREFERRED_ALGORITHM (preferred
 *    DEC algorithm) the standard OpenVMS password hash algorithm’s
 *    are called.
 *
 *    If “alg” is > UAF$C_PREFERRED_ALGORITHM, the following algorithms
 *    are used:
 *    alg == 128: SHA256
 *    alg == 129: A dummy algorithm
 *
 *    To reset an algorithm for a user record in sysuaf.dat, use the
 *    “add/algorithm” or “modify/algorithm” qualifier in the
 *    authorize utility.
 *
 * Input Argument:
 *    pwd_desc - The password as a descriptor
 *    alg - The algorithm to be used.
 *    salt - The salt
 *    usnam_desc - The user name as a descriptor
 * Output Argument:
 *    hashValue - The location where hash is stored
 * Returns:
 *    status
 *
 * The algorithm index is an unsigned byte. Algorithm numbers 0-127 are
 * reserved to HP; customers may use numbers 128-255.
int sys$hash_password (    void *pwd_descr,
    unsigned char alg,
    unsigned short int salt,
    void *usrname_descr,
    struct _generic_64 *hash)
{
    int status;

    #ifdef __HASH_TRACE
    unsigned char *hashBuffer = (unsigned char*) hash;
        tr_print(("In sys$hash_password()"));
    #endif

    /* Establish condition handler, so that any ACCVIO will
       * be returned and not signalled
    */
    lib$establish(EXE$SIGTORET);

    /* Probe the hash buffer for writing */
    status = probe_hash_write (hash);
    if (SS$_NORMAL != status)
    {
        #ifdef __HASH_TRACE
            tr_print(("probe on hash buffer failed: %X", status));
        #endif
        return status;
    }

    /* Zero fill the hash */
    memset (hash, 0, 8);

    /* IF password entry is NULL, return Normal, with zero hash */
    if (NULL == pwd_descr)
    {
        #ifdef __HASH_TRACE
            tr_print(("NULL password descriptor");
        #endif
        return SS$_NORMAL;
    }

    /* Probe the other arguments for reading */
    status = probe_read_args (pwd_descr, username descr);
    if (SS$_NORMAL != status)
    {
        #ifdef __HASH_TRACE
            tr_print(("probe on password/username descriptor failed: %X", status));
        #endif
        return status;
    }

    if (alg <= UAI$C_PREFERRED_ALGORITHM)
    {
        /* OpenVMS preferred algorithm */
        #ifdef __HASH_TRACE
            tr_print(("Calling lgi$hash_password...");
        #endif
        status = lgi$hash_password (pwd_descr, alg, salt,
            username descr, hash);
    }
    else
    {
/* Custom algorithm implementation */
switch (alg)
{
    case 128: /* Using SHA256 */
        #ifdef __HASH_TRACE
            tr_print(("SHA256 algorithm..."));
        #endif
        status = cust$sha256hash_password (pwd_descr, salt,
            usrname_descr, (unsigned char *)hash);
        break;
    case 129: /* Dummy algorithm */
        #ifdef __HASH_TRACE
            tr_print(("Dummy algorithm..."));
        #endif
        status = cust$dummyhash_password (pwd_descr, salt,
            usrname_descr, (unsigned char *)hash);
        break;
    default:
        #ifdef __HASH_TRACE
            tr_print(("Bad algorithm: %X", alg));
        #endif
        status = SS$_BADPARAM;
        break;
} /* End switch (alg) */
} /* End if */

#ifdef __HASH_TRACE
    tr_print (("The hash is %X:%X:%X:%X:%X:%X:%X:%X",
        hashBuffer[0], hashBuffer[1], hashBuffer[2], hashBuffer[3],
        hashBuffer[4], hashBuffer[5], hashBuffer[6], hashBuffer[7]));
    tr_print (("return status: %X", status));
#endif

return status;

/\* The following lines must be included at the end
*\ (Don’t move them to the beginning)
*/
#define INIT001_ROUTINE     hash_password_ext_init
#include <init_rtn_setup.h>

/\* END OF HASH_PASSWORD.C */

SHA2.C differences
SHA2.C was obtained from FREEBSD code from
www.freebsd.org/cgi/cvsweb.cgi/src/sys/crypto/sha2/

The revision "1.9.10.2.1.2.1" of SHA2.C was used and following modification was done to code on OpenVMS to compile:

Changes from:
#include <sys/cdefs.h>
__FBSDID("$FreeBSD: src/sys/crypto/sha2/sha2.c,v 1.10.2.1.2.1 2011/11/11 04:20:22 kensmith Exp "$);

#include <sys/types.h>
#include <sys/time.h>
#else
#include <sys/systm.h>
#endif
#include <string.h>
#endif
#include <machine/endian.h>
#include <crypto/sha2/sha2.h>

To:
#ifndef __VMS
#include <sys/cdefs.h>
__FBSDID("$FreeBSD: src/sys/crypto/sha2/sha2.c,v 1.10.2.1.2.1 2011/11/11 04:20:22
kensmith Exp $");

#include <sys/types.h>
#include <sys/time.h>
#endif
#else
#define assert(x)
#define bcopy(src, dest, siz) memcpy(dest, src, siz)
#define bzero(ptr, siz) memset(ptr, 0, siz)
#define LITTLE_ENDIAN 1234
#define BYTE_ORDER LITTLE_ENDIAN
#endif /* ifdef VMS */

#include <sys/types.h>
#include "sha2.h"
extern void *memcpy (void *dest, const void *source, size_t size);
extern void *memset (void *s, int value, size_t size);
#ifndef __VMS
#define assert(x)
#define bcopy(src, dest, siz) memcpy(dest, src, siz)
#define bzero(ptr, siz) memset(ptr, 0, siz)
#define LITTLE_ENDIAN 1234
#define BYTE_ORDER LITTLE_ENDIAN

Note:
During transfer of the file, the carriage return and line feed (CRLF) might go
wrong, rendering the file unreadable on OpenVMS. We have to properly copy
and use ASCII-mode FTP wherever required.
SHA2.H differences
SHA2.H was obtained from FreeBSD code from www.freebsd.org/cgi/cvsweb.cgi/src/sys/crypto/sha2/.
The revision "1.2.36.1.8.1" of SHA2.H was used and following modification was done to code on OpenVMS to compile:

Changes from:
#if 0
typedef unsigned char u_int8_t;    /* 1-byte (8-bits) */
typedef unsigned int u_int32_t;    /* 4-bytes (32-bits) */
typedef unsigned long long u_int64_t; /* 8-bytes (64-bits) */
#endif

To:
/* #if 0 */
typedef unsigned char u_int8_t;    /* 1-byte (8-bits) */
typedef unsigned int u_int32_t;    /* 4-bytes (32-bits) */
typedef unsigned long long u_int64_t; /* 8-bytes (64-bits) */
/* #endif */

Note:
During transfer of the file, the CRLF might go wrong, rendering the file unreadable on OpenVMS. We have to properly copy and use ASCII mode FTP wherever required.

HASH_PASSWORD_BLD.COM
$!
$! Compile and Link procedure for SYS$HASH_PASSWORD.EXE
$!
$! Compile files
$!
$ CC/PREFIX=ALL/LIST/MACHINE/INSTRUCTION=NOFLOAT -HASH_PASSWORD.C + SYS$LIBRARY:SYS$LIB_C/LIBRARY
$ CC/PREFIX=ALL/LIST/MACHINE/INSTRUCTION=NOFLOAT -SHA2.C + SYS$LIBRARY:SYS$LIB_C/LIBRARY
$!
$! Now link. The SYS$LIBRARY:VMS_EXECEL_LINK.OPT will be available on OpenVMS systems.
$!
Sample build/run

Building

$ show system/noprocess
OpenVMS V8.4 on node SPRING 6-MAR-2012 10:42:15.04  Uptime 1 01:54:06
$ cc /version
HP C V7.3-018 on OpenVMS IA64 V8.4
$ dir
Directory SYS$SYSDEVICE:[PRASADSG.HASH_PASSWORD]
 HASH_PASSWORD.C;1   HASH_PASSWORD.OPT;1 HASH_PASSWORD_BLD.COM;1
 SHA2.C;2            SHA2.C;1            SHA2.H;2            SHA2.H;1

Total of 7 files.
$ @HASH_PASSWORD_BLD.COM
$ dir .exe/date/size
Directory SYS$SYSDEVICE:[PRASADSG.HASH_PASSWORD]
SYS$HASH_PASSWORD.EXE;1
          87   6-MAR-2012 10:42:27.11

Total of 1 file, 87 blocks.
$

Loading (From system account)

$ SET DEFAULT SYS$SYSDEVICE:[PRASADSG.HASH_PASSWORD]
$ COPY SYS$HASH_PASSWORD.EXE SYS$COMMON:[SYS$LDR]SYS$HASH_PASSWORD.EXE
$ MC SYSMAN
SYSMAN> SYS_LOADABLE ADD _LOCAL_ -
SYSMAN> SYS$HASH_PASSWORD -
SYSMAN> /LOAD_STEP = SYSINIT -
SYSMAN> /SEVERITY = WARNING -
SYSMAN> /MESSAGE = "Failure to load SYS$HASH_PASSWORD.EXE"
SYSMAN> EXIT
$ @SYS$UPDATE:VMS$SYSTEM_IMAGES.COM

Reboot the system, so that the new SYS$HASH_PASSWORD.EXE gets loaded. Watch for any errors during reboot on the console.

After rebooting, check if the SYS$HASH_PASSWORD.EXE is really unloaded by doing:

$ analyze/system
OpenVMS system analyzer
SDA> show image SYS$HASH_PASSWORD
Image SYS$HASH_PASSWORD
-----------------------
VMS Executive image layout
--------------------------
<table>
<thead>
<tr>
<th>Image</th>
<th>Length</th>
<th>Image Offset</th>
<th>SymVec</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SYS$HASH_PASSWORD</td>
<td>Data (read/write)</td>
<td>FFFFFFFF.881EDA00</td>
<td>FFFFFFFF.881EDA13</td>
</tr>
<tr>
<td>00000000.00000014</td>
<td>00000000.00010000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00000000.00000004</td>
<td>00000000.00014000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Code</td>
<td></td>
<td>FFFFFFFF.8022FC00</td>
<td>FFFFFFFF.8023500F</td>
</tr>
<tr>
<td>00000000.000005410</td>
<td>00000000.00018000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00000000.000000468</td>
<td>00000000.00020000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Data (read only)</td>
<td></td>
<td>FFFFFFFF.881EDC00</td>
<td>FFFFFFFF.881EDC03</td>
</tr>
<tr>
<td>00000000.00000458</td>
<td>00000000.00024000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Data (read/write)</td>
<td></td>
<td>FFFFFFFF.881EDE00</td>
<td>FFFFFFFF.881EE267</td>
</tr>
<tr>
<td>00000000.00000014</td>
<td>00000000.00028000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Short data (read/write)</td>
<td></td>
<td>FFFFFFFF.881EEA00</td>
<td>FFFFFFFF.881EA13</td>
</tr>
<tr>
<td>00000000.000001D8</td>
<td>00000000.00034000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Linked</td>
<td></td>
<td>FFFFFFFF.881EEC00</td>
<td>FFFFFFFF.881EDD7</td>
</tr>
<tr>
<td>6-MAR-2012 10:42</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GP</td>
<td></td>
<td>FFFFFFFF.883EEC00</td>
<td></td>
</tr>
</tbody>
</table>

SDA> EXIT
$

Modify sample user account
$ SET DEFAULT SYS$SYSTEM
$ MC AUTHORIZE
_UAF> MODIFY -
_UAF> PRASADSG/ALGORITHM=CURRENT=CUSTOMER=128
%UAF-I-MDFYMSG, user record(s) updated
%UAF-W-ALGNEWPWD, password(s) must be reset due to algorithm change
_UAF> MODIFY PRASADSG/PASSWORD=samplepassword
%UAF-I-MDFYMSG, user record(s) updated
_UAF> EXIT
%UAF-I-DONEMSG, system authorization file modified
%UAF-I-NAFNOMODS, no modifications made to network proxy database
%UAF-I-RDNOMODS, no modifications made to rights database
$

Sample login (Telnet) and password reset
$ TELNET 127.0.0.1
%TELNET-I-TRYING, Trying ... 127.0.0.1
%TELNET-I-SESSION, Session 01, host 127.0.0.1, port 23
-TELNET-I-ESCAPE, Escape character is ^]

Welcome to HP OpenVMS Industry Standard 64 Operating System, Version V8.4

Username: prasadsg
Password:
HP OpenVMS Industry Standard 64 Operating System, Version V8.4 on node SPRING
Last interactive login on Tuesday, 6-MAR-2012 10:39:18.84
Last non-interactive login on Tuesday, 6-MAR-2012 10:41:00.70

Your password has expired; you must set a new password to log in

New password:
Verification:
Sample login (SSH) and password reset

$ ssh prasadsg@127.0.0.1

Welcome to HP OpenVMS Industry Standard 64 Operating System, Version V8.4
Prasadsg’s password:

Authentication successful.

Last interactive login on Tuesday, 6-MAR-2012 12:01:10.41
Last non-interactive login on Tuesday, 6-MAR-2012 10:41:00.70

Sample login (set host) and password reset

From console -

$ @sys$startup:tcpi$pshutdown.com

%TCPIP-I-INFO, TCP/IP Services shutdown beginning at 6-MAR-2012 12:09:38.44
-
-
$ set host 0

Welcome to HP OpenVMS Industry Standard 64 Operating System, Version V8.4

Username: prasadsg
Password:

HP OpenVMS Industry Standard 64 Operating System, Version V8.4 on node SPRING
Last interactive login on Tuesday, 6-MAR-2012 12:03:16.54
Last non-interactive login on Tuesday, 6-MAR-2012 10:41:00.70

$
Un-loading (From system account)

**Note:**
Unloading removes the new SYS$HASH_PASSWORD.EXE executable.
After unloading the new SYS$HASH_PASSWORD.EXE:
- All password hash request will be made to the OpenVMS native SYS$HASH_PASSWORD() system service.
- User algorithms should be reverted back to the standard hash algorithm using the authorize utility
- The user passwords need to be reset.

$ MC SYSMAN
SYSMAN> SYS_LOADABLE REMOVE _LOCAL_ SYS$HASH_PASSWORD
SYSMAN> EXIT
$
$ @SYS$UPDATE:VMS$SYSTEM_IMAGES.COM
$
$ @SYS$SYSTEM:SHUTDOWN.COM

SHUTDOWN -- Perform an Orderly System Shutdown
on node SPRING

Do you really want to shutdown node SPRING [NO]? YES

*After rebooting, check if the SYS$HASH_PASSWORD.EXE is really unloaded by doing:*

$ ANALYZE/SYSTEM

OpenVMS system analyzer

SDA> SHOW IMAGE SYS$HASH_PASSWORD
Image SYS$HASH_PASSWORD
-----------------------
No image matching "SYS$HASH_PASSWORD" found
SDA> EXIT

Remove the SYS$COMMON:[SYS$LDR]SYS$HASH_PASSWORD.EXE after the reboot.

$ DELETE SYS$COMMON:[SYS$LDR]SYS$HASH_PASSWORD.EXE;*

**Trouble shooting/Is the new hash really working?**

To troubleshoot or to ensure that the new hash is really working, TR tracing is provided in the HASH_PASSWORD.C file. To enable tracing, rebuild by defining the "__HASH_TRACE" define.

That is, Modify, HASH_PASSWORD_BLD.COM and add "/DEFINE=(_HASH_TRACE)" to the compilation of HASH_PASSWORD.C as follows:

$ CC/PREFIX=ALL/LIST/MACHINE/INSTRUCTION=NOFLOAT/DEFINE=(_HASH_TRACE) -
  HASH_PASSWORD.C + SYS$LIBRARY:SYS$LIB_C/LIBRARY

Rebuild the SYS$HASH_PASSWORD.EXE and place this in SYS$COMMON:[SYS$LDR].

$ @HASH_PASSWORD_BLD.COM
  -
$ COPY SYS$HASH_PASSWORD.EXE SYS$COMMON:[SYS$LDR]SYS$HASH_PASSWORD.EXE
$ DIR SYS$COMMON:[SYS$LDR]SYS$HASH_PASSWORD.EXE
Directory SYS$COMMON:[SYS$LDR]
SYS$HASH_PASSWORD.EXE;2  SYS$HASH_PASSWORD.EXE;1

Total of 2 files.

Reboot the system. After reboot, you can see the TR tracing logs:

$ ANALYZE/SYSTEM

OpenVMS system analyzer

SDA> TR LOAD
TR$DEBUG load status = 00000001
SDA> TR START TRACE
Tracing started...
SDA> EXIT
$ TELNET 127.0.0.1
%TELNET-I-TRYING, Trying ... 127.0.0.1
%TELNET-I-SESSION, Session 01, host 127.0.0.1, port 23
-TELNET-I-ESCAPE, Escape character is ^]

Welcome to HP OpenVMS Industry Standard 64 Operating System, Version V8.4

Username: system
Password:
    HP OpenVMS Industry Standard 64 Operating System, Version V8.4 on node SPRING
    Last interactive login on Tuesday, 6-MAR-2012 12:35:07.48
    Last non-interactive login on Monday, 5-MAR-2012 15:51:43.73
%GNAT_STARTUP-I-SETTING_UP, setting up GNAT
%GNU_STARTUP-I-SETTING_UP, setting up Emacs version 19.28
$ log
    SYSTEM logged out at 6-MAR-2012 12:36:20.68
%TELNET-S-REMCLOSED, Remote connection closed
-TELNET-I-SESSION, Session 01, host 127.0.0.1, port 23
$ $ $ TELNET 127.0.0.1
%TELNET-I-TRYING, Trying ... 127.0.0.1
%TELNET-I-SESSION, Session 01, host 127.0.0.1, port 23
-TELNET-I-ESCAPE, Escape character is ^]

Welcome to HP OpenVMS Industry Standard 64 Operating System, Version V8.4

Username: prasadsg
Password:
    HP OpenVMS Industry Standard 64 Operating System, Version V8.4 on node SPRING
    Last interactive login on Tuesday, 6-MAR-2012 12:35:34.52
    Last non-interactive login on Tuesday, 6-MAR-2012 10:41:00.70
$ $ $ log
    PRASADSG logged out at 6-MAR-2012 12:36:40.52
%TELNET-S-REMCLOSED, Remote connection closed
-TELNET-I-SESSION, Session 01, host 127.0.0.1, port 23
$ $
$ ANALYZE/SYSTEM

OpenVMS system analyzer

SDA> TR SHOW TRACE
Debug Trace Information:
------------------------
<table>
<thead>
<tr>
<th>Timestamp</th>
<th>CPU Buffer</th>
</tr>
</thead>
<tbody>
<tr>
<td>6-MAR 12:36:37.194725</td>
<td>01 return status: 00000001</td>
</tr>
<tr>
<td>6-MAR 12:36:37.194711</td>
<td>01 SHA256 algorithm...</td>
</tr>
<tr>
<td>6-MAR 12:36:37.194710</td>
<td>01 usnarm text addr: 003071B4, Length:8</td>
</tr>
<tr>
<td>6-MAR 12:36:37.194708</td>
<td>01 Pwd text addr: 00011DC4, Length:10</td>
</tr>
<tr>
<td>6-MAR 12:36:37.194702</td>
<td>01 In sys$hash_password()</td>
</tr>
<tr>
<td>6-MAR 12:36:18.149222</td>
<td>00 return status: 00000001</td>
</tr>
<tr>
<td>6-MAR 12:36:18.149220</td>
<td>00 The hash is 00000016:000000E1:000000D1:00000093:0000008F:0000006F:0000006:000000B5</td>
</tr>
<tr>
<td>6-MAR 12:36:18.149175</td>
<td>00 Calling lgi$hash_password...</td>
</tr>
<tr>
<td>6-MAR 12:36:18.149174</td>
<td>00 usnarm text addr: 003071B4, Length:6</td>
</tr>
<tr>
<td>6-MAR 12:36:18.149172</td>
<td>00 Pwd text addr: 00011DC4, Length:10</td>
</tr>
<tr>
<td>6-MAR 12:36:18.149167</td>
<td>00 In sys$hash_password()</td>
</tr>
</tbody>
</table>

SDA> TR STOP TRACE
Tracing stopped...
SDA> TR UNLOAD
TR$DEBUG unload status = 00000001
SDA> EXIT
$

References

1. The OpenVMS Programming Concept Manual: “Implementing Site-Specific Security Policies” has the concept of replacing SYS$HASH_PASSWORD() System service and serves as a good starting place to read. However, the HASH_PASSWORD.MAR referred in the document is only available on OpenVMS VAX systems.

Appendix

License of SHA2.C and SHA2.H

* Copyright 2000 Aaron D. Gifford. All rights reserved.

* Redistribution and use in source and binary forms, with or without
* modification, are permitted provided that the following conditions
* are met:
* 1. Redistributions of source code must retain the above copyright
*    notice, this list of conditions and the following disclaimer.
* 2. Redistributions in binary form must reproduce the above copyright
*    notice, this list of conditions and the following disclaimer in the
*    documentation and/or other materials provided with the distribution.
* 3. Neither the name of the copyright holder nor the names of contributors
*    may be used to endorse or promote products derived from this software
*    without specific prior written permission.

* THIS SOFTWARE IS PROVIDED BY THE AUTHOR(S) AND CONTRIBUTOR(S) “AS IS” AND
* ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
* ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHOR(S) OR CONTRIBUTOR(S) BE LIABLE
* FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
* DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
* OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
* HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
* LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
* OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
* SUCH DAMAGE.