XUSHSH

Robust Hashing
for
ACCESS/VERIFY CODE

pbkdf2, scrypt, sha3

code and documentation
available on github:

https://github.com/grapaZ/xushsh

JohnLeo Zimmer, MD
johnleozim@gmail.com

legacy sign-on security

distribution	`$$EN^XUSHSH("test")`	hash
FOIA	test	“NONE”
OpenVista	test	“NONE”
WorldVistA	115116101116	“LEGACY”
vxVistA	098F6BCD4621D373CADE4E832627B4F6	md5

The Interface to Security

- Weak security that's easy to use will help more people than strong security that's hard to use.
  e.g.: door locks.
- User interface is more important than security.
- Bad user interfaces drive people away from security.
- Weak security is much better than none at all.
  - Rob Pike 2001

- VOE VistA performs an ASCII transform to turn the user's entry into a numeric string. This is not a hash because the process is easily reversed to retrieve the original text.
- VistA invokes the routine ^XUSHSH to obfuscate each user's ACCESS CODE and VERIFY CODE when stored in the NEW PERSON FILE in global ^VA(200,

^XUSHSH
Legacy “hash”

```
EN(X) ; GENERIC HASHING ENCRYPTION --
USES ASCII ENCODING
 N %HASH S %HASH=""
 N %CHAR
 F %CHAR=1:1:$L(X) D
 . I %CHAR#2 S %HASH=$A(X,%CHAR)_%HASH
 . E  S %HASH=%HASH_$A(X,%CHAR)
 Q %HASH
```

```
MU-beta>w $$EN^XUSHSH("test")
 115116101116
```

unhash

```
UN(X) ;TMG/kst - UWINDS FOIA ASCII encoding
      New %HASH Set %HASH=""
      New %TMP Set %TMP=""
      New %DIGIT Set %DIGIT=""
      New I
      For I=1:1:$L(X) Do
      . Set %DIGIT=%DIGIT_$E(X,I)
      . If (+%DIGIT>31) Do
      . . Set %TMP=%TMP_$char(%DIGIT)
      . . Set %DIGIT=""
      For I=$L(%TMP):-1:1 Do
      . If I#2 Do
      . . Set %HASH=$E(%TMP,1)_%HASH    ;"get 1st char
      . . Set %TMP=$E(%TMP,2,$L(%TMP))  ;"trim off 1st char
      . else  Do
      . . Set %HASH=$E(%TMP,$L(%TMP))_%HASH ;"get last char
      . . Set %TMP=$E(%TMP,1,$L(%TMP)-1)    ;"trim last char
      Quit %HASH
```

```
MU-beta>w $$UN^VWHSH0("115116101116")
 test
```

ACCESS CONTROL needs a real hash

Earlier work

- Chris Uyehara
  - SHA1 written in MUMPS
  - Call to openssl -sha512
- DSS/LM
  - Md5 written in MUMPS

Evolving security environment

Evolving security environment

All three still useful... but not so much for password protection. Attacks have evolved in both hardware (e.g. GPU) and software (e.g. rainbow tables).

- Md5 weakest
- SHA1 obsolete
- SHA512 stronger, but still ***TOO FAST!***

Sometimes when you fill the vacuum,
it still sucks. Rob Pike

properties of ideal cryptographic hash:

- Can compute the hash value for any given input
- Cannot find an input yielding a specified hash
- Cannot modify input without changing the hash
- Cannot find two inputs with the same hash.

properties of ideal cryptographic hash:

- Can compute the hash value for any given input
- Cannot find an input yielding a specified hash
- Cannot modify input without changing the hash
- Cannot find two inputs with the same hash.

- Costly to calculate *(and adjustable)*...

Demonstration of effect of iterations on time to compute pbkdf2 log-on hash

($$ZHOROLOG^%POSIX)

iterations	seconds	hash of "test"
00001	0.037	579bb89e389d45611e735d85c926b20ea0276d2fee5c5b95
00010	0.027	ffb03d6ff7cfadbb99ee72885522e66209956bf32e37c458
00100	0.048	d7a0e3b279505ebf37fb3783060d8120cc86616755be7b06
01000	0.087	ce979774611b26b29d2760eb522a47254a51d392729ad657
10000	0.55	9e27047241e3f9bc8d03bc1f590ff7792d5b32713faeca9d
20000	1.08	a8661f93452892f8e3d6249173fb59911e45b8b7e2adcc37
30000	1.62	8e8086478433afe834b6a9b3b723e00f77fe0b3f7d32b3af
40000	2.18	a8616f276e2f7064c2d39a39372710a727cef4007e671384
50000	2.79	48b104d53e33fea3e21be6b216bc44613cad0f1af481c97c
60000	3.29	2984ee4b89341d34d7fde9dd17f9f5a7daef67dcb87859b3
70000	3.96	cffccdd28e7e6a146b7d9c3d09360ddb45754eaf62873a32
80000	4.55	5e6c28eb7ff0b6f57f6dfa94af67e0079a95bbc8f47a7b69
90000	5.12	fb423815326261679a6f92a2ee49bbc16746eed3aa22db98

100000 3.95 ← pbkdf2.py uses recursion and segfaults after about 93,540 iterations

properties of ideal cryptographic hash:

- Can compute the hash value for any given input
- Cannot find an input yielding a specified hash
- Cannot modify input without changing the hash
- Cannot find two inputs with the same hash.

- Costly to calculate...

of both CPU and memory

Effect of iterations (N) and memory (r, p) on time to compute scrypt hashes

N	r	p	seconds
16	1	1	.081707
1024	16	8	.241603
16384	1	8	.247368
1048576	1	8	11.540036
2097152	8	1	24.236959

```
scrypt$1024 16 8 16$saltydog$12a5862a1ba3cbee789bb388c427842f
```

```
GTM>s X="SM1234!!"

GTM>S p="pipe" O p:(COMMAND="sed -e 's/$/\r/' | gpg --print-md SHA256"):: "PIPE"

GTM>Use p Write X,! Write /EOF Read Y Close p

GTM>W !,X,!,Y

SM1234!!

58D079D3 613DD04F AD6F092C 600E92DE 83D2323D 545CF3BB 6994464C 97C8E5CD
```

```
XUSHSH ;JL.Z; MORE ROBUST PASSWORD ENCRYPTION ; 5 SEPTEMBER 2009
 ;; GT.M VERSION
 ;; Depends on gpg for hash
 ;; Copyright 2009 JohnLeo Zimmer
 ;;
 ;;
 ;;SF-ISC/STAFF - PASSWORD ENCRYPTION ;3/23/89  15:09 ; 4/14/05 1:22pm
 ;; Input in X
 ;; Output in X
 ;; Algorithm for VistA Office EHR encryption (BSL)
 ;;
A ;
 S X=$$EN(X)
 Q
 ;
EN(X,HASH) ; SHA-256 hash
 IF $G(HASH)="" S HASH="SHA256" ;; DEFAULT is SHA256
 IF HASH'["SHA" Q X             ;; EN(X,"something else") returns unhashed X
 OPEN "PIPE":(COMMAND="sed -e 's/$/\r/' | gpg --print-md "_HASH)::"PIPE"
 USE "PIPE" W X,! W /EOF R X
 CLOSE "PIPE"
 QUIT X
```

```
XUSHSH ;;GpZ;  ; IMPROVED HASHING UTILITY: Cache/Windows
               (VWHSHCWN);01/08/2010
   V ;;8.0;KERNEL;;Jul 10, 1995
    ;;
   A S X=$$EN(X) Q
    ;;
EN(X,HASH) ;;
    N (X,HASH)
    D:'$L($G(^%ZOSF("HASHLIST"))) DEFHASH^VWHSH0
    S HASHLIST=^%ZOSF("HASHLIST")
    S HASH=$S('$L($G(HASH)):
$P(HASHLIST,"|",1),1:$TR(HASH,"abcdefghijklmnopqrstuvwxyz-
","ABCDEFGHIJKLMNOPQRSTUVWXYZ"))
    IF HASH="LEGACY" QUIT $$EN^VWHSHLEG(X)
    Q:HASHLIST'[(" "_HASH_" ") X
    N PIPE,ZUT,I
    S ZUT=$ZUTIL(68,40,1)
    S PIPE=" echo "_X_"||"_$P(HASHLIST,"||",3)_"gpg --print-md
"_HASH
    OPEN PIPE:"Q"
    F I=1:1:4 USE PIPE R X Q:$ZEOF<0  S HASHOUT=$G(HASHOUT)_X
    CLOSE PIPE
    S ZUT=$ZUTIL(68,40,ZUT),X=HASHOUT
    Q $TR(X," ")
```

```
XUSHSH ;;GpZ; - ; IMPROVED HASHING UTILITY: for Cache/Linux (VWHSHCLX);
                  01/08/2010
    V ;;8.0;KERNEL;;Jul 10, 1995
     ;;
    A S X=$$EN(X) Q
     ;;
    EN(X,HASH) ;;
     N (X,HASH)
     D:'$L($G(^%ZOSF("HASHLIST"))) DEFHASH^VWHSH0
     S HASHLIST=^%ZOSF("HASHLIST")
     S HASH=$S('$L($G(HASH)):$P(HASHLIST,"|",1),1:$TR(HASH,
          "abcdefghijklmnopqrstuvwxyz- ","ABCDEFGHIJKLMNOPQRSTUVWXYZ"))
     IF HASH="LEGACY" QUIT $$EN^VWHSHLEG(X)
     Q:HASHLIST'[(" "_HASH_" ") X
     S SED="sed -e 's/$/\r/'|"
     N PIPE,ZUT,I
     S ZUT=$ZUTIL(68,40,1)  ;; MSM-style End-of-File Handling
     S PIPE=" echo "_X_"||"_SED_$P(HASHLIST,"||",3)_"gpg --print-md "_HASH
     OPEN PIPE:"Q"
     F I=1:1:4 USE PIPE R X Q:$ZEOF<0  S HASHOUT=$G(HASHOUT)_X
     CLOSE PIPE
     S ZUT=$ZUTIL(68,40,ZUT),X=HASHOUT
     Q $TR(X," ")
```

```
XUSHSH ;;GpZ; - ; IMPROVED HASHING UTILITY: GT.M Version (VWHSHGTM);
                  01/08/2010
   V ;;8.0;KERNEL;;Jul 10, 1995
    ;;
   A S X=$$EN(X) Q
    ;;
   EN(X,HASH) ;;
    N (X,HASH)
    D:'$L($G(^%ZOSF("HASHLIST"))) DEFHASH^VWHSH0
    S HASHLIST=^%ZOSF("HASHLIST")
    S HASH=$S('$L($G(HASH)):$P(HASHLIST,"|",1),1:$TR(HASH,
      "abcdefghijklmnopqrstuvwxyz-
      ","ABCDEFGHIJKLMNOPQRSTUVWXYZ"))
    IF HASH="LEGACY" QUIT $$EN^VWHSHLEG(X)
    Q:HASHLIST'[(" "_HASH_" ") X
    S SED="sed -e 's/$/\r/'|"
    OPEN "PIPE":(COMM=SED_"gpg --print-md "_HASH)::"PIPE"
    USE "PIPE" W X,! W /EOF
    F  R X Q:$ZEOF  S HASHOUT=$G(HASHOUT)_X
    CLOSE "PIPE"
    Q $TR(HASHOUT," ")
```

SHA-3 (Keccak)
pysha3 for Python 2.6 – 3.4

```
c:\Python27>python
Python 2.7.5 (default, May 15 2013, 22:44:16) [MSC v.1500 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import sys
>>> import hashlib
>>> if sys.version_info < (3, 4):
...     import sha3
...
>>> s = hashlib.new("sha3_512")
>>> s =hashlib.sha3_512()
>>> s.name
'sha3_512'
>>> s.digest_size
64
>>> s.update(b"data")
>>> s.hexdigest()
'1065aceeded3a5e4412e2187e919bffeadf815f5bd73d37fe00d384fe29f55f08462fdabe1007b9
93ce5b8119630e7db93101d9425d6e352e22ffe3dcb56b825'
>>>
```

https://pypi.python.org/pypi/pysha3/0.3

SHA-3 (Keccak)

```
Import sha3


# sha3_228(), sha3_256(), sha3_384(), and sha3_512().

if (hsh=="sha3"):
    if (keylen < 48):
          hash=hashlib.sha3_256   # 32
    elif (keylen > 63):
          hash=hashlib.sha3_512   # 64
    else:
          hash=hashlib.sha3_384   # 48
    Flag=1


if (flag==1):
     print hsh+"$"+salt+"$"+hash(input +
salt).hexdigest()
     exit()
```

- Calls to `$$EN^XUSHSH`
  - `XUS`
  - `USERBLK`
  - `XUSBE1`
  - `XUS2`
  - `XOBSRAKJ`

- Calls to `$$CHECKAV^XUS`
  - `XMRPOP`
  - `XUSG`
  - `XUSRA`
  - `XUSRB`
  - `XUSRB5`
  - `XUVERIFY`

CHECKAV^XUS(X1)

```
.
.
.
S X=$P(X1,";") Q:X="^" -1 S:XUF %1="Access: "_X
Q:X'?1.20ANP 0
S X=$$EN^XUSHSH(X) I '$D(^VA(200,"A",X)) D LBAV Q 0
S %1="",IEN=$O(^VA(200,"A",X,0)),XUF(.3)=IEN D USER(IEN)
S X=$P(X1,";",2) S:XUF %1="Verify: "_X S X=$
$EN^XUSHSH(X)
I $P(XUSER(1),"^",2)'=X D LBAV Q 0
.
.
.
```

prepare MUMPS environment:

- We are now ready to hook these tools into the MUMPS code of the Kernel. Our distribution provides a configuration utility, `^VWHSH0` will be used to prepare the ground and then to install the proper version tailored to either GT.M or Cache. A configuration array stored at `^VA(200,"VWHSH")` is built, ready to redirect control to the chosen hash algorithm. Final conversion of the NEW PERSON file is done, in Stage 3 by calls into `^VWHSH0`.
- routine entry points
- VWHSH0 TEST, SAVEOLD, BUILD(), MOVEIN
- TO(FROMHASH,TOHASH), REVERT, KILL
- VWHSH3 Caché version XUSHSH
- VWHSH8 GT.M version XUSHSH

https://github.com/grapaZ/xushsh

Readme.txt

robust_xushsh.odt/.pdf

XUSHSH

Robust Hashing
for
ACCESS/VERIFY CODE

pbkdf2, scrypt, sha3

code and documentation
available on github:

https://github.com/grapaZ/xushsh

JohnLeo Zimmer, MD
johnleozim@gmail.com

legacy sign-on security

distribution	`$$EN^XUSHSH("test")`	hash
FOIA	test	“NONE”
OpenVista	test	“NONE”
WorldVistA	115116101116	“LEGACY”
vxVistA	098F6BCD4621D373CADE4E832627B4F6	md5

The Interface to Security

- Weak security that's easy to use will help more people than strong security that's hard to use.
  e.g.: door locks.
- User interface is more important than security.
- Bad user interfaces drive people away from security.
- Weak security is much better than none at all.

  – Rob Pike 2001

- VOE VistA performs an ASCII transform to turn the user's entry into a numeric string. This is not a hash because the process is easily reversed to retrieve the original text.
- VistA invokes the routine ^XUSHSH to obfuscate each user's ACCESS CODE and VERIFY CODE when stored in the NEW PERSON FILE in global ^VA(200,

^XUSHSH
Legacy “hash”

```
EN(X) ; GENERIC HASHING ENCRYPTION --
USES ASCII ENCODING
 N %HASH S %HASH=""
 N %CHAR
 F %CHAR=1:1:$L(X) D
 . I %CHAR#2 S %HASH=$A(X,%CHAR)_%HASH
 . E  S %HASH=%HASH_$A(X,%CHAR)
 Q %HASH
```

```
MU-beta>w $$EN^XUSHSH("test")
 115116101116
```

unhash

```
UN(X) ;TMG/kst - UWINDS FOIA ASCII encoding
      New %HASH Set %HASH=""
      New %TMP Set %TMP=""
      New %DIGIT Set %DIGIT=""
      New I
      For I=1:1:$L(X) Do
      . Set %DIGIT=%DIGIT_$E(X,I)
      . If (+%DIGIT>31) Do
      . . Set %TMP=%TMP_$char(%DIGIT)
      . . Set %DIGIT=""
      For I=$L(%TMP):-1:1 Do
      . If I#2 Do
      . . Set %HASH=$E(%TMP,1)_%HASH    ;"get 1st char
      . . Set %TMP=$E(%TMP,2,$L(%TMP))  ;"trim off 1st char
      . else  Do
      . . Set %HASH=$E(%TMP,$L(%TMP))_%HASH ;"get last char
      . . Set %TMP=$E(%TMP,1,$L(%TMP)-1)    ;"trim last char
      Quit %HASH
```

```
MU-beta>w $$UN^VWHSH0("115116101116")
 test
```

ACCESS CONTROL
needs a real hash

Earlier work

- Chris Uyehara
  - SHA1 written in MUMPS
  - Call to openssl -sha512
- DSS/LM
  - Md5 written in MUMPS

Evolving security environment

First, please do not use SHA-1. It is quite vulnerable
http://eurocrypt2009rump.cr.yp.to/837a0a8086fa6ca714249409ddfae43d.pdf
and Federal agencies must phase out its use by the end of 2010
http://csrc.nist.gov/groups/ST/hash/statement.html

Secondly, in general, and for cryptographic functions specifically, it is preferable to use standard external libraries. It is better to use heavily used, widely scrutinized code that will be regularly patched than specific, ad hoc code. Both MUMPS implementations used for VistA provide straightforward access to standard external libraries, and that is IMHO the right way to compute cryptographic hashes.

Regards
-- Bhaskar 6/12/2009

Evolving security environment

All three still useful... but not so much for password protection. Attacks have evolved in both hardware (e.g. GPU) and software (e.g. rainbow tables).

- Md5 weakest
- SHA1 obsolete
- SHA512 stronger, but still ***TOO FAST!***

```
GTM>s X="SM1234!!"

GTM>S p="pipe" O p:(COMMAND="sed -e 's/$/\r/' | gpg --print-md
SHA256"):::"PIPE"

GTM>Use p Write X,! Write /EOF Read Y Close p

GTM>W !,X,!,Y

SM1234!!

58D079D3 613DD04F AD6F092C 600E92DE 83D2323D 545CF3BB
6994464C 97C8E5CDj
```

Sometimes when you fill the vacuum,
it still sucks. Rob Pike

properties of ideal cryptographic hash:

- Can compute the hash value for any given input
- Cannot find an input yielding a specified hash
- Cannot modify input without changing the hash
- Cannot find two inputs with the same hash.

properties of ideal cryptographic hash:

- Can compute the hash value for any given input
- Cannot find an input yielding a specified hash
- Cannot modify input without changing the hash
- Cannot find two inputs with the same hash.

- Costly to calculate *(and adjustable)*...

Demonstration of effect of iterations on time to compute pbkdf2 log-on hash

($$ZHOROLOG^%POSIX)

iterations	seconds	hash of "test"
00001	0.037	579bb89e389d45611e735d85c926b20ea0276d2fee5c5b95
00010	0.027	ffb03d6ff7cfadbb99ee72885522e66209956bf32e37c458
00100	0.048	d7a0e3b279505ebf37fb3783060d8120cc86616755be7b06
01000	0.087	ce979774611b26b29d2760eb522a47254a51d392729ad657
10000	0.55	9e27047241e3f9bc8d03bc1f590ff7792d5b32713faeca9d
20000	1.08	a8661f93452892f8e3d6249173fb59911e45b8b7e2adcc37
30000	1.62	8e8086478433afe834b6a9b3b723e00f77fe0b3f7d32b3af
40000	2.18	a8616f276e2f7064c2d39a39372710a727cef4007e671384
50000	2.79	48b104d53e33fea3e21be6b216bc44613cad0f1af481c97c
60000	3.29	2984ee4b89341d34d7fde9dd17f9f5a7daef67dcb87859b3
70000	3.96	cffccdd28e7e6a146b7d9c3d09360ddb45754eaf62873a32
80000	4.55	5e6c28eb7ff0b6f57f6dfa94af67e0079a95bbc8f47a7b69
90000	5.12	fb423815326261679a6f92a2ee49bbc16746eed3aa22db98

100000 3.95 ← pbkdf2.py uses recursion and segfaults after about 93,540 iterations

properties of ideal cryptographic hash:

- Can compute the hash value for any given input
- Cannot find an input yielding a specified hash
- Cannot modify input without changing the hash
- Cannot find two inputs with the same hash.

- Costly to calculate...

  of both CPU and memory

Effect of iterations (N) and memory (r, p) on time to compute scrypt hashes

N	r	p	seconds
16	1	1	.081707
1024	16	8	.241603
16384	1	8	.247368
1048576	1	8	11.540036
2097152	8	1	24.236959

scrypt$1024 16 8 16$saltydog$12a5862a1ba3cbee789bb388c427842f

```
demo    ;;2014-01-19  10:54 AM

    for mod="16 1 1 16","1024 8 16 16","16384 8 1 16","16384 16 2
16","1048576 8 1 16","2097152 8 1 16" do
    . ;write !,mod,!
    . set START=$Piece($$ZHOROLOG^%POSIX,",",2)
    . ;; write "START "_START,!
    . write $$OS^VWHSH8("python /home/vista/bin/xus.py -z
'scrypt$"_mod_"$saltydog'"),"    "
    . set STOP=$Piece($$ZHOROLOG^%POSIX,",",2)
    . ;; write "STOP "_STOP,!
    . write STOP-START,!
    . quit
```

```
GTM>s X="SM1234!!"

GTM>S p="pipe" O p:(COMMAND="sed -e 's/$/\r/' | gpg --print-md SHA256"):"PIPE"

GTM>Use p Write X,! Write /EOF Read Y Close p

GTM>W !,X,!,Y

SM1234!!

58D079D3 613DD04F AD6F092C 600E92DE 83D2323D 545CF3BB 6994464C 97C8E5CD
```

```
XUSHSH ;JL.Z; MORE ROBUST PASSWORD ENCRYPTION ; 5 SEPTEMBER 2009
 ;; GT.M VERSION
 ;; Depends on gpg for hash
 ;; Copyright 2009 JohnLeo Zimmer
 ;;
 ;;
 ;;SF-ISC/STAFF - PASSWORD ENCRYPTION ;3/23/89  15:09 ; 4/14/05 1:22pm
 ;; Input in X
 ;; Output in X
 ;; Algorithm for VistA Office EHR encryption (BSL)
 ;;
A ;
 S X=$$EN(X)
 Q
 ;
EN(X,HASH) ; SHA-256 hash
 IF $G(HASH)="" S HASH="SHA256" ;; DEFAULT is SHA256
 IF HASH'["SHA" Q X             ;; EN(X,"something else") returns unhashed X
 OPEN "PIPE":(COMMAND="sed -e 's/$/\r/' | gpg --print-md "_HASH)::"PIPE"
 USE "PIPE" W X,! W /EOF R X
 CLOSE "PIPE"
 QUIT X
```

Lars Nooden lars.curator@gmail.com via googlegroups.com
9/5/09

to hardhats
JohnLeo Zimmer wrote:
> Thanks, Lars
> This version may be a bit cleaner in a gt.m pipe.

You're welcome. It is only a matter of removing occurrences of ctrl-M,
when paired with ctrl-J, and so can be even more generic using sed:

```
sed 's/\x0D$//' < test | gpg --print-md SHA256
```

dos2unix is available everywhere, but not by default on some systems.

```
XUSHSH ;;GpZ;  ; IMPROVED HASHING UTILITY: Cache/Windows
               (VWHSHCWN);01/08/2010
   V ;;8.0;KERNEL;;Jul 10, 1995
    ;;
   A S X=$$EN(X) Q
    ;;
EN(X,HASH) ;;
    N (X,HASH)
    D:'$L($G(^%ZOSF("HASHLIST"))) DEFHASH^VWHSH0
    S HASHLIST=^%ZOSF("HASHLIST")
    S HASH=$S('$L($G(HASH)):
$P(HASHLIST,"|",1),1:$TR(HASH,"abcdefghijklmnopqrstuvwxyz-
","ABCDEFGHIJKLMNOPQRSTUVWXYZ"))
    IF HASH="LEGACY" QUIT $$EN^VWHSHLEG(X)
    Q:HASHLIST'[(" "_HASH_" ") X
    N PIPE,ZUT,I
    S ZUT=$ZUTIL(68,40,1)
    S PIPE=" echo "_X_"||"_$P(HASHLIST,"||",3)_"gpg --print-md
"_HASH
    OPEN PIPE:"Q"
    F I=1:1:4 USE PIPE R X Q:$ZEOF<0  S HASHOUT=$G(HASHOUT)_X
    CLOSE PIPE
    S ZUT=$ZUTIL(68,40,ZUT),X=HASHOUT
    Q $TR(X," ")
```

```
XUSHSH ;;GpZ; - ; IMPROVED HASHING UTILITY: for Cache/Linux (VWHSHCLX);
                  01/08/2010
    V ;;8.0;KERNEL;;Jul 10, 1995
     ;;
    A S X=$$EN(X) Q
     ;;
    EN(X,HASH) ;;
     N (X,HASH)
     D:'$L($G(^%ZOSF("HASHLIST"))) DEFHASH^VWHSH0
     S HASHLIST=^%ZOSF("HASHLIST")
     S HASH=$S('$L($G(HASH)):$P(HASHLIST,"|",1),1:$TR(HASH,
          "abcdefghijklmnopqrstuvwxyz- ","ABCDEFGHIJKLMNOPQRSTUVWXYZ"))
     IF HASH="LEGACY" QUIT $$EN^VWHSHLEG(X)
     Q:HASHLIST'[(" "_HASH_" ") X
     S SED="sed -e 's/$/\r/'|"
     N PIPE,ZUT,I
     S ZUT=$ZUTIL(68,40,1)  ;; MSM-style End-of-File Handling
     S PIPE=" echo "_X_"||"_SED_$P(HASHLIST,"||",3)_"gpg --print-md "_HASH
     OPEN PIPE:"Q"
     F I=1:1:4 USE PIPE R X Q:$ZEOF<0  S HASHOUT=$G(HASHOUT)_X
     CLOSE PIPE
     S ZUT=$ZUTIL(68,40,ZUT),X=HASHOUT
     Q $TR(X," ")
```

```
XUSHSH ;;GpZ; - ; IMPROVED HASHING UTILITY: GT.M Version (VWHSHGTM);
                  01/08/2010
   V ;;8.0;KERNEL;;Jul 10, 1995
    ;;
   A S X=$$EN(X) Q
    ;;
   EN(X,HASH) ;;
    N (X,HASH)
    D:'$L($G(^%ZOSF("HASHLIST"))) DEFHASH^VWHSH0
    S HASHLIST=^%ZOSF("HASHLIST")
    S HASH=$S('$L($G(HASH)):$P(HASHLIST,"|",1),1:$TR(HASH,
      "abcdefghijklmnopqrstuvwxyz-
      ","ABCDEFGHIJKLMNOPQRSTUVWXYZ"))
    IF HASH="LEGACY" QUIT $$EN^VWHSHLEG(X)
    Q:HASHLIST'[(" "_HASH_" ") X
    S SED="sed -e 's/$/\r/'|"
    OPEN "PIPE":(COMM=SED_"gpg --print-md "_HASH)::"PIPE"
    USE "PIPE" W X,! W /EOF
    F  R X Q:$ZEOF  S HASHOUT=$G(HASHOUT)_X
    CLOSE "PIPE"
    Q $TR(HASHOUT," ")
```

SHA-3 (Keccak)
pysha3 for Python 2.6 – 3.4

```
c:\Python27>python
Python 2.7.5 (default, May 15 2013, 22:44:16) [MSC v.1500 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import sys
>>> import hashlib
>>> if sys.version_info < (3, 4):
...     import sha3
...
>>> s = hashlib.new("sha3_512")
>>> s =hashlib.sha3_512()
>>> s.name
'sha3_512'
>>> s.digest_size
64
>>> s.update(b"data")
>>> s.hexdigest()
'1065aceeded3a5e4412e2187e919bffeadf815f5bd73d37fe00d384fe29f55f08462fdabe1007b9
93ce5b8119630e7db93101d9425d6e352e22ffe3dcb56b825'
>>>
```

https://pypi.python.org/pypi/pysha3/0.3

SHA-3 (Keccak)

```
Import sha3


# sha3_228(), sha3_256(), sha3_384(), and sha3_512().

if (hsh=="sha3"):
    if (keylen < 48):
          hash=hashlib.sha3_256   # 32
    elif (keylen > 63):
          hash=hashlib.sha3_512   # 64
    else:
          hash=hashlib.sha3_384   # 48
    Flag=1


if (flag==1):
     print hsh+"$"+salt+"$"+hash(input +
salt).hexdigest()
     exit()
```

- Calls to `$$EN^XUSHSH`
  - `XUS`
  - `USERBLK`
  - `XUSBE1`
  - `XUS2`
  - `XOBSRAKJ`

- Calls to `$$CHECKAV^XUS`
  - `XMRPOP`
  - `XUSG`
  - `XUSRA`
  - `XUSRB`
  - `XUSRB5`
  - `XUVERIFY`

CHECKAV^XUS(X1)

```
.
.
.
S X=$P(X1,";") Q:X="^" -1 S:XUF %1="Access: "_X
Q:X'?1.20ANP 0
S X=$$EN^XUSHSH(X) I '$D(^VA(200,"A",X)) D LBAV Q 0
S %1="",IEN=$O(^VA(200,"A",X,0)),XUF(.3)=IEN D USER(IEN)
S X=$P(X1,";",2) S:XUF %1="Verify: "_X S X=$
$EN^XUSHSH(X)
I $P(XUSER(1),"^",2)'=X D LBAV Q 0
.
.
.
```

prepare MUMPS environment:

- We are now ready to hook these tools into the MUMPS code of the Kernel. Our distribution provides a configuration utility, `^VWHSH0` will be used to prepare the ground and then to install the proper version tailored to either GT.M or Cache. A configuration array stored at `^VA(200,"VWHSH")` is built, ready to redirect control to the chosen hash algorithm. Final conversion of the NEW PERSON file is done, in Stage 3 by calls into `^VWHSH0`.
- routine entry points
- VWHSH0 TEST, SAVEOLD, BUILD(), MOVEIN
- TO(FROMHASH,TOHASH), REVERT, KILL
- VWHSH3 Caché version XUSHSH
- VWHSH8 GT.M version XUSHSH

https://github.com/grapaZ/xushsh

Readme.txt

robust_xushsh.odt/.pdf