

Smart Card HOTP

Open Source HMAC-Based One-Time
Password Algorithm Implementation for *nix and
RADIUS capable hardware.

Agenda

- Two Factor Authentication with HOTP.
- Smart Cards & Readers.
- Authentication Hooks into servers.
- Admin tools for HOTP management.
- Installation and Provisioning walkthrough.

HOTP

- Definition and security analysis of algorithm in RFC 4226.
- Authors members of the Open AuTHentication Initiative OATH.
- Open standard two factor authentication algorithm.
- Compete with existing vendor proprietary solutions.

RFC 4226:

- “The absence of open specifications has led to solutions where hardware and software components are tightly coupled through proprietary technology, resulting in high-cost solutions, poor adoption, and limited innovation.”

Two Factor Authentication

- Something you have -- a hardware token generator, printed token list, magnetic key card, RFID card, etc
- Something you know -- a password.
- Both are required for successful authentication.

Given the opportunity...

- Easy to guess passwords are chosen.
- Passwords will be shared among systems creating transitive security relationships.
- Passwords will be cached, ssh included (Windows ssh client clickbox).
- Policy discouraging the above is ineffective.

When a password is not working...

- Try your “other one”
- Then the “other other one”
- Common occurrence when password changes notices are given.
- Your router and VPN passwords just got sent over an insecure link to a hacked web site storing {srcIP, username, password} combinations.

HOTP

- Each token only works once.
- Immune to replaying of captured, guessed, or cracked passwords used by end users.
- Token is computationally infeasible to generate without access to key material (typically) not revealed to end users.

Working system

- End user token generator.
- Authentication back end for systems requiring HOTP logins.
- Administrative tools to manage keys, reset a user PIN, program smart cards and reader.
- Misc -- scripts to bulk configure users.

Existing solutions

- Commercial and open source options available
- S/KEY (open source)
- RSA Secure ID (proprietary)
- Many others which fall somewhere between open source and proprietary.

Commercial Vendors

- Secure-ID, Crypto Card, others.
- Geared toward enterprise deployments, not easy to integrate into corner cases such as a lone terminal server at a remote site on a dialup modem, or other one-off's. Our Policies require OTP everywhere. It only takes one compromised backbone router...

S/KEY One-Time Password System

- Open Source, Open Standard RFC 2289.
- Widely implemented for *nix.
- Somewhat unique in that server does not share any key material with the token generator.

S/KEY One-Time Password System

- A good hardware token generator not available.
- Does not work well for non technical users.
- Carrying a printed list for every system requiring S/KEY access is not practical.
- End user habits can easily compromise security.

S/KEY One-Time Password System

1: MID JURY ODD ALGA SAW YAM
2: ODIN GUT SLY COW CLAM LEER
3: EMIT COMB SAIL SEEK FIEF TON
4: DEFT ME OMEN MOO IOWA EROS
5: SWAG AGEE SORT BOMB BELT RIG
6: ALSO EASE NERO WIRE BAD TED
7: LIAR TOY MAD ROAR SOUL MILD
8: ABUT GYP THAT DEL MASH SHOT
9: KURD RAIN RYE JUG AWK BURG
10: ROBE MUCH ALVA IDEA DARN RULE

Wishlist

- No dependencies on network based authentication servers -- we must be able to get into our equipment during maintenance and outages without special procedures. Equipment is physically located at many sites across the state.
- * No client software -- too many compatibility and support issues.

Wishlist

- Easy to integrated into a variety of hardware / authentication systems.
- Open algorithm, not locked into a single vendor.
- Palatable to staff, low nuisance factor.
- Low cost for small deployments, ie single server with a few users.

New Requirements

- Autonomy across groups (OARnet Engineering, OARnet systems, OSC HPC).
- No dependencies across groups, HPC not dependent on OARnet.
- Open source the implementation. Try to avoid being a poorly documented unmaintained local hack.

Issues not addressed by OTP's

- Login must be encrypted and mutually authenticated (ssh/https) or session can be hijacked. OTP is not a replacement for strong encryption.
- End user workstation must be secured. Logging in from a hacked PC nullifies OTP security. No remote logins, servers, etc on staff workstations / laptops.

Issues not addressed by OTP's

- Poor key management with ssh and HTTPS has trained end users to ignore warnings indicating possible hijacked sessions.
- Hijacking a session (MIM attack) was once considered difficult. No longer with compromised end-user WIFI and broadband routers.

HOTP Algorithm

- $HOTP(K,C) = \text{Truncate}(HMAC\text{-}SHA\text{-}1(K,C))$
- K is a secret key.
- C is a count which increases on each successful authentication.
- K and C are shared between the token generator (end user) and auth server.

HMAC-SHA-1

- Keyed-Hashing for Message Authentication.
- Defined in FIPS PUB 198 (US Department of Commerce, Technology Administration, NIST)
- Apply a secure hash to a Key and a Message (Counter for HOTP).
- The result is the One Time Password.



In English...

- The Token requires a secret key and not secret count to generate.
- The key can not be derived from the Token except by brute force which is computationally infeasible.
- When the count changes (successful authentication) the next token will also change (OTP).

In English...

- Once a count value has been used it is never valid again. The server must keep track of “old” count values by increasing its copy of the count.
- Without the shared secret a valid token can not be generated.
- End users should not have access to the key material. The token generator must be tamper resistant.

Implications...

- The count must be synchronized between the token generator and server.
- For optimum security the key should be unique per user, per system. An end user configured to login to multiple servers will have multiple (logical) token generators. Some cases where this restriction can be relaxed.

Implications...

- There must be a way for a token generator to re-synchronize to the server. If the user generates a token and then does not use it the count will not match the server resulting in a failed authentication. The user could manually enter the count, or the server could “look ahead” for small synch errors.

HOTP Algorithm

- $\text{HOTP}(K,C) = \text{Truncate}(\text{HMAC-SHA-1}(K,C))$
- HMAC-SHA-1 produces 160 bits of output, this is asking too much of someone to type in. `Truncate()` reduces the output, to say 40 bits which can be represented as 10 HEX nybbles.

HOTP Algorithm

- Truncating to 40 bits greatly reduces the security of the OTP. The server must implement an authentication rate limiting mechanism to prevent exhaustive token space attacks. A limit of one authentication per second results in about 17,000 years for a 50% probability of hitting the 40 bit token (good enough).

Smart Cards

- Contact and Contactless (RFID)
- EEPROM (memory card, dumb device)
- Microcontroller, Flash, RAM (MCU card)
- JAVA capable cards (high end MCU type)
- Purpose programmed cards:
PKSC#11, .NET Sun Ray...

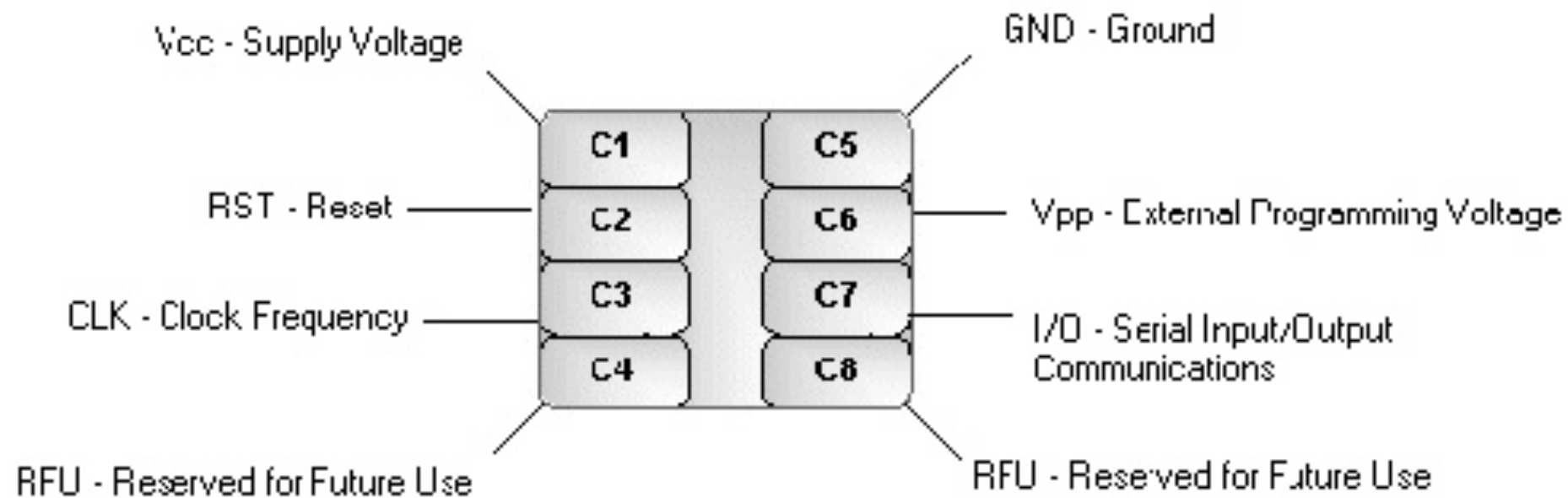
Smart Cards

- ISO-7816-* defines physical characteristics, electrical signals and protocols.
- PC/SC API for interfacing software through a smart card reader to a smart card.
- Broad range of capabilities depending on vendor and model.

Smart Cards



Electrical Contacts





BasicCard

- ZC 3.9 \$1.50 in single quantity. 256 bytes of RAM and 8K EEPROM.
- Free Development IDE with crypto library (Windows Based).
- Sufficient resources for HOTP implementation with 85 stored systems.



Balance Reader

- Low Cost (\$10 single quantity).
- Used for E-Cache applications.
- Popular in the UK.
- Insert Smart Card, HOTP generated.
- Limited to a single system (HOTP key).
- No provisions for user input such as a PIN or challenge request.

Stand-Alone Reader

- Spyrus PAR II \$60 single quantity.
- PIC16F877 Microcontroller 14K program memory, 368 bytes RAM, 256 bytes EEPROM.
- Co-Controller to offload LCD, keypad, and smart card interface.
- Allows for PIN, multiple systems menu, challenge entry.



Demo Reader



- PIN: 12345
- ↓ menu activate
- * change PIN
- # toggle challenge prompt
- <enter> Generate HOTP token
- nn shortcut to system

PC Based Readers

- \$15 single quantity.
- USB or Serial.
- Different form factors.
- Reader == Reader/Writer. Same interface. For MCU based cards, readers are all alike. For proprietary formats like EEPROM reader may need to be paired with Smart Card.



Smart Card Firmware

- T=1 (Block) Async Protocol
- CLA,INS (Class, Instruction) - Function
- Data block - In/Out data

Smart Card Firmware

```
#ifdef ENABLECGETVERSION  
command &H80 &H52 GetVersion(V as Byte)  
  
    V = HOTPCodeVersion  
  
end command  
#endif 'ENABLECGETVERSION
```


Smart Card Firmware

```
#ifdef ENABLECGETHOSTNAME
command &H80 &H44 GetHostName(Idx as Byte, myPIN as String*5, _
    HostName as String*12)

    if CheckPIN(myPIN) <> 0 then
        SW1SW2 = swAccessDenied
        Exit
    end if

    if CheckIndex(Idx) <> 0 then
        SW1SW2 = swDataNotFound
        Exit
    end if

    HostName = HOTPHost(idx)

end command
#endif 'ENABLECGETHOSTNAME
```


Smart Card Terminal

```
scrio.tx_lc = ZC_VERSION_LEN;
scrio.tx_le = ZC_VERSION_LEN;
scrio.tx_delay = ZC_GETVERSION_DELAY;
scrio.rx_buf_len = scrio.tx_le;
i = 0;

scrio.tx_buf[i++] = ZC_GETVERSION_CLA;      /* CLA */
scrio.tx_buf[i++] = ZC_GETVERSION_INS;     /* INS */
scrio.tx_buf[i++] = 0x0;                   /* P1 */
scrio.tx_buf[i++] = 0x0;                   /* P2 */
scrio.tx_buf[i++] = scrio.tx_lc;           /* LC */
for (j = 0; j < ZC_VERSION_LEN; ++j)
    scrio.tx_buf[i++] = 0;
scrio.tx_buf[i++] = scrio.tx_le;           /* LE */

scrio.tx_buf_len = i;
```


Smart Card Terminal

```
/* SC transaction */
if (scr_ctx_cmd(scrctx, &scrio) < 0) {
    if (scrctx->verbose)
        xerr_warnx("sc_ctx_cmd(): failed.");
    return -1;
}

/* check response code and size */
if ((r = scr_checks1sw2_rx(&scrio, scrio.tx_le, 0x90, 0x00,
    scrctx->verbose)) != 0) {
    return -1;
}

/* copy out SC response */
for (j = 0; j < ZC_VERSION_LEN; ++j)
    zc_version[j] = scrio.rx_buf[j];
```


Smart Card HOTP

- Non volatile storage for n systems
{Hostname, Shared Key, Count}
- Administrative commands for storing and retrieving systems, reset user PIN
- User commands to generate a HOTP token, verify & reset PIN, list system names.

otp-sct

- Command line smart card terminal program.
- Same functionality of Spyrus reader when smart card and reader are connected to PC.
- HOTP generated on the card and presented to the user.
- Reduces security of system.

otp-sct

```
otp-sct [-lhelpv?] [-c count] [-d debug_level] [-i index] [-r reader]
-h : help
-l : list SC readers
-L : list hostnames
-p : reset PIN
-v : list SC firmware version
```


Working system

- End user token generator.
- Authentication back end for systems requiring HOTP logins.
- Administrative tools to manage keys, reset a user PIN, program smart cards and reader.
- Misc -- scripts to bulk configure users.

Back end

- C library implementation of HOTP algorithm.
- Maintain database of {Username, Count, Count Ceiling, Shared Key, Last Authentication Attempt, Account Status}.
- Authentication hooks.

pam_otp.so

- Pluggable Authentication Module (PAM)- de-facto standard for interfacing authentication methods to Unix servers.
- Install OTP library, set PAM configuration to include pam_otp.so in the authentication chain.
- OARnet deployment limited sshd.

urd

- Micro RADIUS daemon for authentication with OTP library.
- Allows VPN appliance, routers, switches, to authenticate with HOTP.

otp-openvpn

- OpenVPN Authentication Plug-In program for interfacing OpenVPN with HOTP.
- OpenVPN has some issues with OTP's in general....

C API

```
if (!(otpctx = otp_db_open(otpdb_fname, db_flags)))
    xerr_errx(1, "otp_db_open(): failed.");

if ((r = otp_user_exists(otpctx, username)) < 0)
    xerr_errx(1, "otp_user_exists(): failed.");

if (r != 0)
    xerr_errx(1, "User %s does not exist in otp database.", username);

if ((r = otp_user_auth(otpctx, username, pass, OTP_HOTP_WINDOW)) < 0)
    xerr_errx(1, "otp_user_auth(): failed.");

if (otp_db_close(otpctx) < 0)
    xerr_errx(1, "otp_user_close(): failed.");

/*
 * r == OTP_AUTH_PASS then pass authentication
 * else fail.
 */
```


Working system

- End user token generator.
- Authentication back end for systems requiring HOTP logins.
- Administrative tools to manage keys, reset a user PIN, program smart cards and reader.
- Misc -- scripts to bulk configure users.

otp-control

- otp-control maintains the back-end user database.
- Database is stored as ASCII files in a format similar to `passwd(5)`.
- add, remove, modify, test key.
- Requires escalated privileges to r/w the OTP database. Not necessarily root.

otp-control

```
otp-control [-?hnv] [-c count] [-C count_ceil] [-F sc_flags] [-H sc_hostname]
[-I sc_index] [-k key] [-m command_mode] [-o otbdb_pathname]
[-u username] [-w window]
```

-h : help
-n : create database
-v : enable verbose output

sc_flags : 0=CHALLENGE, 1=READERKEY

Mode	Description
add	- Add user
activate	- Activate user
create	- Create database
deactivate	- Deactivate user
disable	- Disable user
dump	- ASCII dump user record(s)
flags-dspcnt	- Set user display count flag.
flags-no-dspcnt	- Clear user display count flag.
generate	- Generate HOTP for user
list	- List user record (printable)
list-sc	- List user record (SC friendly)
load	- ASCII load user record(s)
remove	- Remove user
set-count	- Reset count for user
set-count-ceil	- Reset count ceiling for user
test	- Test user

otp-sca

- otp-sca maintains the database on the end user smart cards via a PC connected smart card reader.
- add, remove, modify systems.
- Reset a user PIN.
- Test HOTP generation.

otp-sca

- Admin functions require the use of an admin key -- end users can not execute the admin commands.
- Enable/Disable balance card support. Balance cards can not support PIN functionality and may violate local policies.
- Implements other SC commands.

otp-sca

```
otp-sca [-hlp?] [-a admin_keyfile] [-c count] [-d debug_level]
        [-i index] [-m command_mode] [-M modifiers] [-r reader]
        [-R reader_keyfile] [-u username] [-v card_api_version]
```

-h : help
-l : list SC readers
-p : no PIN required

Command Mode	Description	Notes	Modifiers
admin-enable	- Enable Admin Mode	1	
admin-disable	- Disable Admin Mode		
adminkey-set	- Set Admin Key	1	
balancecard-set	- Set Balance Card Index	1	
capabilities-get	- Get Capabilities		
host-get	- Get host entry	1,2,4	d
host-set	- Set host entry	1,4	
hostname-get	- Get Hostname for Index	2,3	
hotp-gen	- Generate HOTP for Index	3	chr
pin-set	- Set PIN	3	
pin-test	- Test/Verify PIN	3	
reader-key-set	- Set Reader Key	1	
sc-clear	- Clear all SC data	1	
spyrus-ee-get	- Spyrus EEPROM read	5	
spyrus-ee-set	- Spyrus EEPROM write	5	
version	- Firmware version		

Notes (*):

- 1 Admin Enable required.
- 2 Iterate over all if no index specified.
- 3 PIN or Admin Enable required.
- 4 version 3 firmware supports 32 bit count, version 2 16 bit count.
- 5 Spyrus customization SC firmware

Modifiers: (version 3+ SC firmware)

- c pass count to SC.
- h return hostname from SC.
- d output in otpdb load friendly format.
- r include reader key in request.

htsoft-downloader

- Spyrus PAR II reader microcontroller (PIC16F877) must have HOTP terminal firmware programmed.
- Required for “new” vendor supplied readers (done once).
- RS232 downloader cable or USB to RS232 adapter for laptops.
- Windows version also available.

bcload

- Zeitcontrol Smart Card microcontroller must have HOTP card firmware programmed.
- Required for “new” vendor supplied cards (done once).
- Zeitcontrol supplied for Windows.
- Local version for Unix.

Working system

- End user token generator.
- Authentication back end for systems requiring HOTP logins.
- Administrative tools to manage keys, reset a user PIN, program smart cards and reader.
- Misc -- scripts to bulk configure users.

users2otpdb otpdb2sc

- In Unix tradition many small programs which do a single task. Scripts used for actual deployment with list of users.
- Add a list of users to OTP db with random keys.
- Pull users from multiple OTP db's and create file ready to dump to smart card.

Working system

- End user token generator.
- Authentication back end for systems requiring HOTP logins.
- Administrative tools to manage keys, reset a user PIN, program smart cards and reader.
- Misc -- scripts to bulk configure users.

Installation & Provisioning Walkthrough

Back End Requirements

- openssl - crypto library
- pcsc-lite - optional open source PC/SC drivers. Driver for ACR30S chipset embedded in application. pcsc-lite adds support for many smart card readers. Used by otp-sct and otp-sca.
- ccid - optional driver package for pcsc-lite.

Back End Requirements

- `acr38u` - optional driver package for `pcsc-lite`.
- `gcc` and `pam-development` not installed by default in some Linux distributions.
- see `QUICKSTART` for package install notes.

Build

- Targets for Mac, FreeBSD, and Linux.
- Not auto tooled yet.

```
# build Intel Linux, pcsc-lite installed with yum
cd otp
cd common; make clean; make i386-yum-linux; cd ..
cd bclload; make clean; make i386-yum-linux; make install; cd ..
cd htsoft-downloader; make clean; make i386-linux; make install; cd ..
cd otp-control; make clean; make i386-linux; make install; cd ..
cd otp-pam; make clean; make i386-linux; make install; cd ..
cd otp-sca; make clean; make i386-yum-linux; make install; cd ..
cd otp-sct; make clean; make i386-yum-linux; make install; cd ..
cd otp-openvpn; make clean; make i386-linux; make install; cd ..
cd urd; make clean; make i386-linux; make install; cd ..
cd basiccard; make install; cd ..
cd spyrus-par2; make install; cd ..
cd scripts; make install; cd ..
```


Create OTP db

```
mkdir /etc/otpdb  
chown root:wheel /etc/otpdb  
chmod 700 /etc/otpdb
```


Install PAM module

```
# linux
cp otp-pam/pam_otp.so /lib/security
chown root:wheel /lib/security/pam_otp.so
chmod 755 /lib/security/pam_otp.so
# freebsd
cp otp-pam/pam_otp.so /usr/lib
chown root:wheel /usr/lib/pam_otp.so
chmod 755 /usr/lib/pam_otp.so
```


Add first user

```
cd otp-control
```

```
./otp-control -n -u joe -m add
```

```
./otp-control -u joe -m deactivate
```

```
./otp-control -u joe -m list
```

```
>Username.....joe
```

```
>Key.....784F37E95A8410400700DF1E52466AB1704F487B
```

```
>Count.....0 (0x0)
```

```
>Count Ceiling..18446744073709551615 (0xFFFFFFFFFFFFFFFF)
```

```
>Version.....1
```

```
>Status.....inactive (2)
```

```
>Format.....hex40 (1)
```

```
>Type.....HOTP (1)
```


Configure PAM/ SSHD

```
/etc/pam.d/sshd:  
# change auth lines:  
auth      requisite      pam_unix.so      nullok try_first_pass  
auth      required       pam_otp.so       expose_account display_count  
allow_inactive debug  
# expose_account enabled verbose logging via syslog:  
#   OTP username=joe response=0E3F8E7C47  
# display_count enables the HOTP count in the challenge prompt  
#   HOTP Challenge (1843):  
#           ^^^^ this is the count  
# allow_inactive will configure the module to allow a user in the OTP  
# database set to status inactive to pass authentication without an  
OTP.
```


Configure PAM/ SSHD

```
/etc/ssh/sshd_config:
```

```
# PasswordAuthentication must be turned off (default is on)  
# (note this is not true for all versions of sshd, see example  
# below.
```

```
# SSH-2.0-OpenSSH_5.2 - PasswordAuthentication yes
```

```
# SSH-2.0-OpenSSH_4.5p1 - PasswordAuthentication no
```

```
PasswordAuthentication no
```

```
# usePAM to yes (default)
```

```
UsePAM yes
```

```
# ChallengeResponseAuthentication is required for the pam OTP module  
# to interact with sshd
```

```
ChallengeResponseAuthentication yes
```

```
# Public Key Authentication must also be turned off
```

```
RSAAuthentication no
```

```
PubkeyAuthentication no
```


Restart sshd

```
# restart sshd (linux)
/etc/init.d/sshd restart
# restart sshd (FreeBSD)
/etc/rc.d/sshd restart
```


sshd behavior...

```
# example of incorrectly configured system, note after 3 attempts
# with PAM, sshd reverts to internal authentication code allowing the
# OTP PAM module to be bypassed.
#
# with later versions of sshd this is no longer true, ie
# SSH-2.0-OpenSSH_5.2 is okay. The second password prompt will
# also call pam_otp
#
bastion.eng:~% ssh 10.1.0.25 -l 'joe'
Password:
Password:
Password:
joe@10.1.0.25's password:
```


Test sshd/OTP

```
# generate OTP
```

```
./otp-control -u joe -m generate  
count=5 crsp=48B0D8D8E1
```

```
# verify sshd is still working properly
```

```
bastion.eng:~% ssh 10.1.0.25
```

```
Password:
```

```
Last login: Tue Sep  1 23:21:20 2009 from 10.1.0.26
```

```
# activate user
```

```
./otp-control -u joe -m activate
```

```
# login with OTP generated earlier
```

```
bastion.eng:~% ssh 10.1.0.25
```

```
Password:
```

```
HOTP Challenge (5): 48B0D8D8E1
```

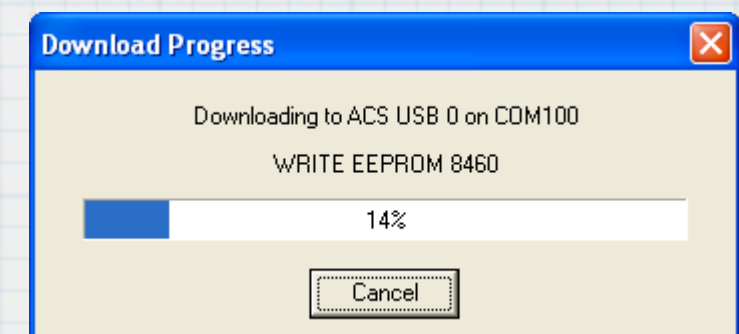
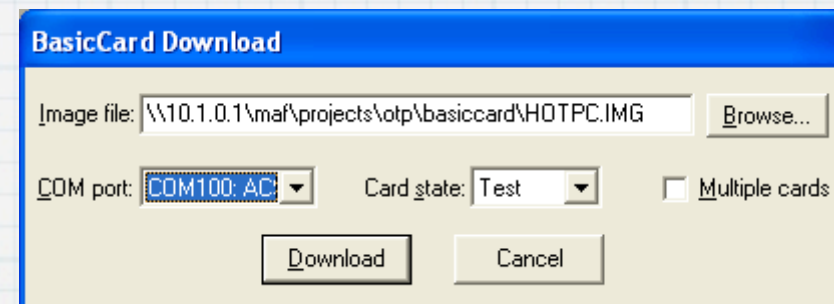
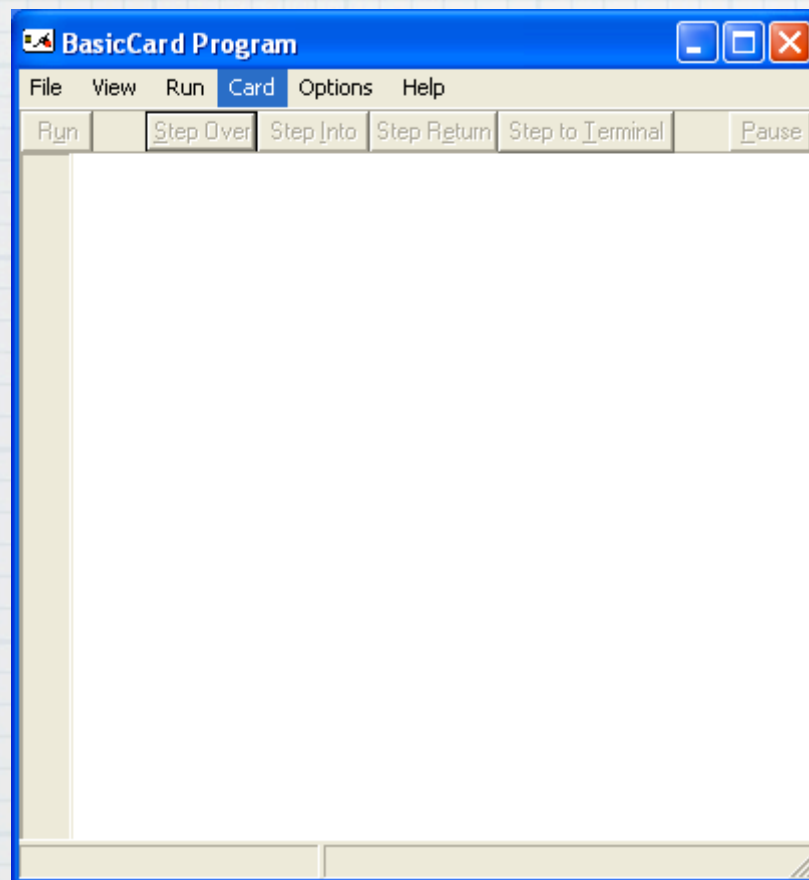
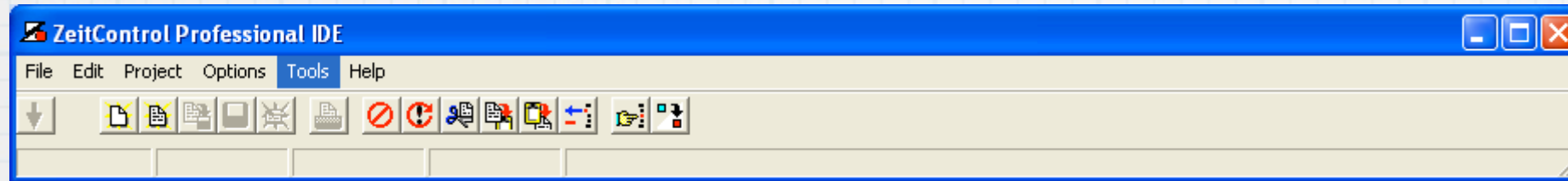
```
Last login: Wed Sep  2 00:22:03 2009 from 10.1.0.26
```

```
[joe@localhost ~]$
```


sshd with OTP and X

- Login to bastion host once with X forwarding enabled.
- Start multiple xterm's forwarded to local display.
- OARnet deployment requires OTP authentication when logging in at start of the day. Minimal nuisance/time commitment.

Load SC firmware (Windows)



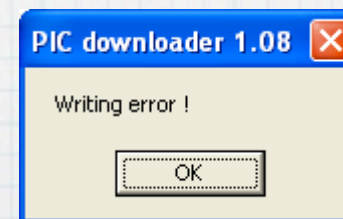
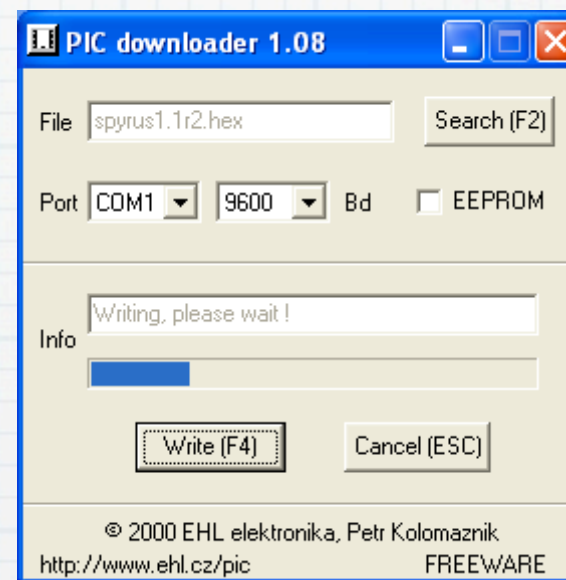
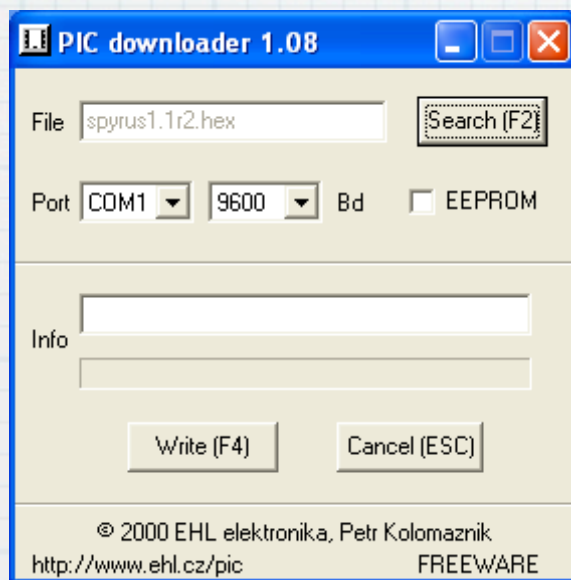
Load SC firmware (Unix)

```
./bclload -v -f $OOTP/firmware/HOTPC.IMG
```

```
Card/State: ZC3.9 test
EEChunkSize=2000
BCSetState: load
SC: Reset
EEStart=8020,EELen=1fa0
imgAddr=8020,imgLen=1fa0
Clear: addr=8020,len=1fa0
BCClearEEProm: success
SC: Reset
EEWRITE: nWrites=121,addr=8020,len=1e
EEWRITE: nWrites=120,addr=8038,len=10
EEWRITE: nWrites=119,addr=8170,len=38
EEWRITE: nWrites=118,addr=81a0,len=48
...
EEWRITE: nWrites=1,addr=9f40,len=48
EEWRITE: nWrites=0,addr=9f80,len=48
EECRC: nWrites=1,addr=8020,len=1fa0,imgCRC=7a3f
EECRC: SCCRC=7a3f
EECRC: nWrites=0,addr=8020,len=00,imgCRC=00
EECRC: SCCRC=0
BCSetState: test
SC: Reset
```



Load Spyrus PAR II Firmware (Windows)



Firmware loads

- Done once for factory supplied Smart Cards and PAR II reader.

Set Smart Card Admin Key

```
./otp-sca -l  
embedded:acr30s  
PCSC:OmniKey CardMan 1021 00 00
```

Enable admin mode with default key:

```
echo "3030303030303030303030303030303030303030303030" >  
default.key  
./otp-sca -m admin-enable -a default.key
```

Create a new admin key with openssl and set the
smart card to use it:

```
openssl rand 160 | openssl sha1 > secret.key  
./otp-sca -a secret.key -m adminkey-set
```


Provision Smart Card

Dump user joe in an otp-sca friendly format:

```
cd ../otp-control  
./otp-control -u joe -m list-sc -I0 -Hcrypto |  
tail -1 > /tmp/joe.bastion.card
```

```
cd ../otp-sca
```

Load the above record into the smart card index 00. The current compiled in limit for a ZC3.9 card is 85 systems:

```
./otp-sca -m host-set < /tmp/joe/bastion.card
```

```
rm /tmp/joe.bastion.list
```

Outside test environment use a pipe instead of temp files for added security.

Verify Provisioning

```
./otp-sca -m host-get
```

```
#index:count:hostname:key  
00:00000000:63727970746F000000000000:784F37E95A841  
0400700DF1E52466AB1704F487B
```

Note the hostname is encoded in hex. The key matches the one programmed above into the otpdb used with PAM. Initial count is set to 0.

Dump the available hosts in a friendlier format. The card is still in admin mode so the PIN does not matter:

```
./otp-sca -m hostname-get  
Enter PIN: 99999  
00,crypto
```


Set PIN and Test

Disable admin mode:

```
./otp-sca -m admin-disable -a secret.key
```

Set PIN for card. The default 28165 PIN can not be used to generate a HOTP:

```
./otp-sca -m pin-set  
Enter PIN: 28165  
New PIN: 12345  
New PIN (again): 12345  
SetPIN Good.
```

Generate a HOTP for user.

```
./otp-sca -m hotp-gen -Mch -c5  
Enter PIN: 12345  
HOTP: 48B0D8D8E1 -- crypto
```


Provisioning

- Admin Key is typically shared for all cards.
- Steps to verify programming, set PIN, test HOTP not normally done. Verbose example.
- Typically all the above steps would be handled by the auto gen script fed with a list of users and host systems.

In production...

```
% ssh dev1.eng.oar.net
```

```
Password:
```

```
HOTP Challenge (1880): ffaf2b77b2
```

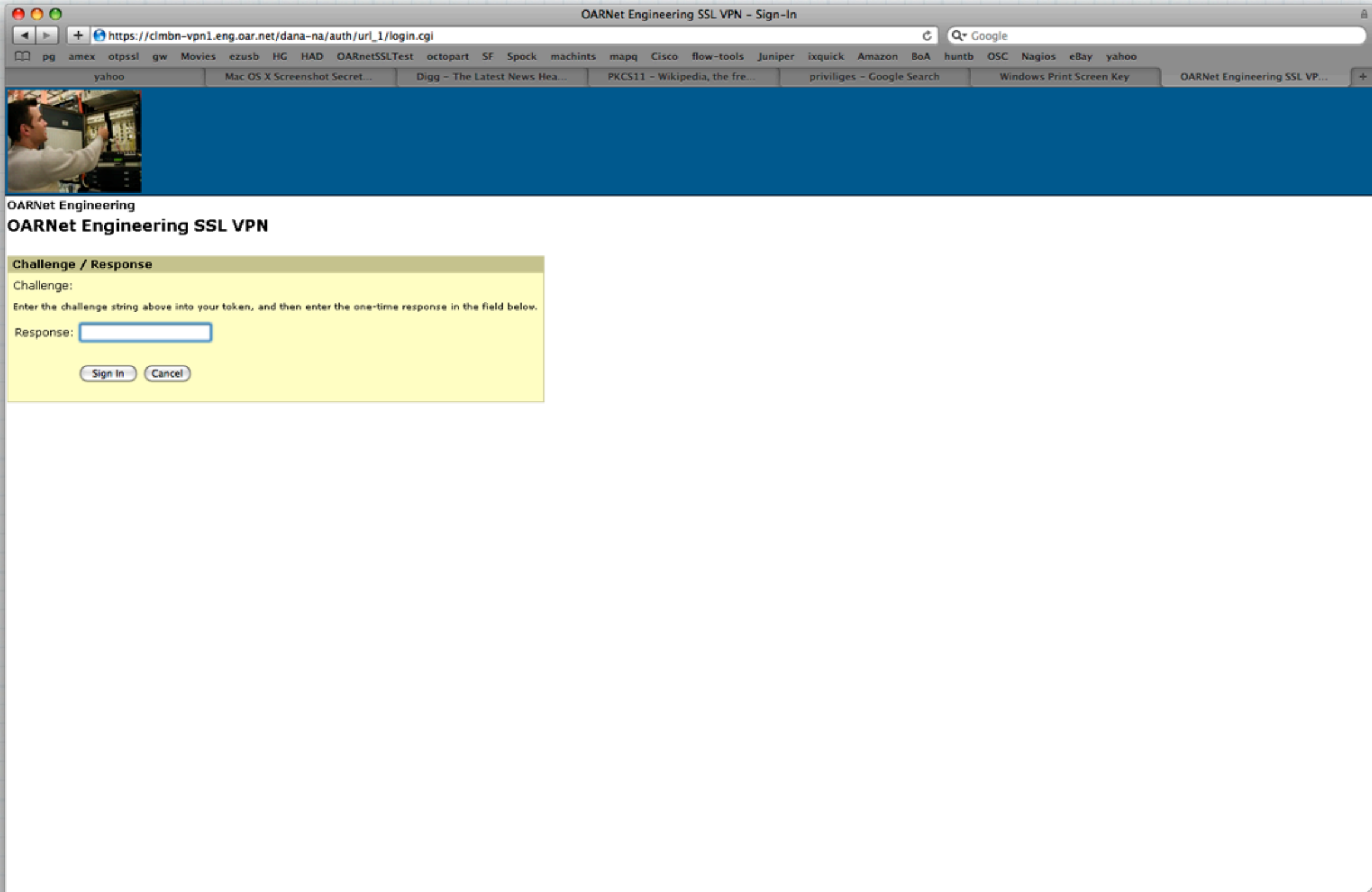
```
Copyright (c) 1980, 1983, 1986, 1988, 1990, 1991, 1993, 1994
```

```
The Regents of the University of California. All rights reserved.
```

```
FreeBSD OARNET_20071228 (GENERIC) #0: Mon Dec 31 05:12:37 UTC 2007
```

```
-----  
This system is for the use of authorized users only. Individuals using  
this computer system without authority, or in excess of their authority,  
are subject to having all of their activities on this system monitored  
and recorded by system personnel. In the course of monitoring individuals  
improperly using this system, or in the course of system maintenance,  
the activities of authorized users may also be monitored. Anyone using  
this system expressly consents to such monitoring and is advised that if  
such monitoring reveals possible evidence of criminal activity, system  
personnel may provide the evidence of such monitoring to law enforcement  
officials.  
-----
```


In production...




OARNet Engineering SSL VPN - Sign-In

https://clmbn-vpn1.eng.oar.net/dana-na/auth/url_1/login.cgi

pg amex otpssl gw Movies ezusb HG HAD OARNetSSLTest octopart SF Spock machints mapq Cisco flow-tools Juniper ixquick Amazon BoA huntb OSC Nagios eBay yahoo

yahoo Mac OS X Screenshot Secret... Digg - The Latest News Hea... PKCS11 - Wikipedia, the fre... privileges - Google Search Windows Print Screen Key OARNet Engineering SSL VP...



OARNet Engineering
OARNet Engineering SSL VPN

Challenge / Response

Challenge:

Enter the challenge string above into your token, and then enter the one-time response in the field below.

Response:

Deployment

- Required/Production on bastion hosts since Feb 2006, testing Jun 2005. Protecting all backbone routers, switches and servers.
- Many recent software changes including RADIUS support, PC/SC support, formal documentation, new features. In testing/review stage.

Implementation security

- Communication between the reader and Smart Card is not encrypted.
- HOTPC.IMG is distributed in test mode, which may allow reading of the BasicCard without the admin key.
- Be careful with PAM and sshd “PasswordAuthentication”. See QUICKSTART

Implementation security

- User PIN is 5 digits. The Smart Card will disable itself after 10 consecutive failed attempts to protect against brute-force attacks.
- The HOTP token is 40 bits in this implementation. To guard against brute-force attacks otplib will rate-limit authentication requests to 1 per second.

Implementation security

- OpenSSL is required for the HMAC implementation.
- libcrypt is required for the RADIUS daemon Unix password authentication.
- pcsc-lite for SC maintenance tools.
pccs-lite is not used in the authentication back-end (ie PAM module).

Implementation security

- The HOTP database is not encrypted. Use an encrypted filesystem if this is a concern. An encrypted fs can also be used for storing the user .card files SC admin key, and READERKEY.
- READERKEY is weak authentication (40 bit shared key between PARII and SC) to discourage use of HOTP generation on PC connected readers.

Deployment

- SSL/VPN Radius support will add OTP protection for Optical network and internal web services.
- Challenge synch option on SpyruS readers allows provisioning of two (or more) smart cards per user. Useful for staff who occasionally forget reader and card at home.

Deployment

- All OARnet engineering managed systems will be behind OTP, including one-off's such as terminal servers on dialup modems to conform with recent security policy changes.
- People forget the cards and reader at home. Latest software will allow staff to have multiple cards. With SpyruS reader user has option to sync count.

Smart Card HOTP

Mark Fullmer
maf@eng.oar.net