CNS Lecture 3

- authentication
- passwords
- hash functions
  - MD5, SHA, RIPEMD-160, Whirlpool
  - fun things you can do with ~~hashish~~ hashing

assignments

```
root:.6PcYNUPBpzxw:0:0::/root:/bin/csh
toor::0:0::/root:/bin/csh
alice:nfVWnUWaAuDSE:500:500:Alice:/home/alice:/bin/csh
mallory:yyv4LsfKaOHAE:503:500:Mallory:/home/mallory:/bin/csh
bob:nfVWnUWaAuDSE:501:500:Bob:/home/bob:/bin/csh
eve:xxP6v.rTrWTb2:505:500:Eve:/home/eve:/bin/csh
```

In the news

- **gcc 4.1 has stackguard**
- **Problem in OpenSSL with SHA public key of 3 and padding**
  CVE-2006-4339 see cve.mitre.org
- **Bank of Ireland will refund phishing losses (160K euros)**
- **Spammer conviction upheld (9 years in prison)**
- **Microsoft Publisher could allow remote execution**

Cost-benefit analysis for the attacker (Clark & Davis '95)

$M_b + P_b > O_{cp} + O_{cm} P_a P_c$

- M_b monetary benefit to attacker
- P_b psychological benefit to attacker
- O_{cp} cost of committing the crime
- O_{cm} cost of conviction to the attacker
- P_a probability of arrest
- P_c probability of conviction

Hacking Tip 7345:
Reverse engineer a "fixed" application or DLL and develop an exploit ☺

You are here …

Attacks & Defenses
- Risk assessment✓
- Viruses✓
- Unix security✓
- authentication
- Network security
  Firewalls,vpn,IPsec,IDS
- Forensics

Cryptography
- Random numbers
- Hash functions
  MD5, SHA,RIPEMD
- Classical + stego
- Number theory
- Symmetric key
  DES, Rijndael, RC5
- Public key
  RSA, DSA, D-H,ECC

Applied crypto
- SSH
- PGP
- S/Mime
- SSL
- Kerberos
- IPsec
- Crypto APIs
- Coding securely

authentication

- verifying an identity
- people authentication
- password schemes
- challenge/response
- token based
- public key systems
- biometrics
- host/message authentication

Why authentication?
- access control
- authorization
- auditing

Design criteria

- strength (resistance to attacks)
- speed
- ease of use
- accuracy (false positives)
- manageability
- reliability

Authenticating people

computer verifying who you are

- what you know
- what you have
- what you are

Best: at least two of the above

people identify people by
- face
- voice
- ID
- reference

Is it you?

What you know
- •password
- •PIN
- •Social security number
- •Mother's maiden name

don't know when you've lost/shared it!

What you are … biometrics
- •facial verification (picture ID)
- •retinal scanner
- •fingerprint reader
- •hand geometry reader (Olympics/customs)
  - verification, not identifcation
- •voiceprints
- •keystroke timing
- •handwriting

concerns: expense, false accept/reject

What you have
- •ATM card
- •dongle (bootup, software protect)
- •SecurID (server, modified login)
- •crypto calculator (challenge/response)
- •smart card/disk

You know when you've lost it

combine with what you know: PIN or signature

special readers / software mods

Smart cards

- memory (cash card)
- secret-key crypto cards
- public-key crypto cards (FORTEZZA)
- need RS-232 reader or PCMCIA or USB or smartdisk
- need card loading/init device
- ISO standards

excellent for storing private keys or biometric data

Provide both logical and physical access control (DoD)

Smart card features

- reader powered (9600 baud)
- tamper-resistant (self-destruct)
- 8-bit CPU, 8KB ROM, 3K EEPROM, RAM
- Write/Erase Cycles: 10,000
- Data life (storage) : 10 years

crypto co-processor (FORTEZZA pcmcia)
- 160-bit CPU
- mod arithmetic
- crypto software (RSA, skipjack)

Smart card readers (RS-232, USB, PCMCIA)
- login supported by linux, Windows

Java Card Specifications

- CPU: 8, 16 bit ‹Micro-controller
- Memory: EEPROM 32k, 64k and (soon) 128k
- External Clock Frequency: 1 to 7.5 MHz
- Operating Temperature: –25 to +75 C
- Data retention: 10 years
- Standards: ISO 7816, Java Card 2.1.1, Open Platform 2.0.1
- Security: DES, Triple DES, RSA 1024, SHA-1, X.509 certificates, On-Card key generation

Authentication scorecard

Total Cost of Ownership	Acquisition Cost	• What are the initial acquisition costs? • Include all additional hardware, software, servers, readers, services, etc. associated with acquiring the authentication solution.
	Deployment Cost	• What are the costs to deploy the authentication solution? • This includes the distribution of any necessary hardware or software; ease of installation; ease of setup and configuration; training of end-users; etc.
	Operating Cost	• What are the ongoing operating costs? • This may include costs for replacement (e.g., expired / lost / stolen / broken) authentication devices; ongoing management; upgrades; vendor support; help desk support; etc.
Strategic Fit (users)	Convenience/ Ease of Use	• What kinds of end-user population(s) will be supported? • How easy is it for end-users to learn how to use the authentication method? • How convenient is it for end-users to use the authentication method, day in and day out?
	Portability	• How portable is the authentication method? • Can it reliably be used to gain access from multiple locations (office, home, airport, hotel, kiosk, etc.)
	Multi-Purpose	• Can the authentication method be used for more than one purpose? e.g., network access, physical access, application access, photo ID badge, electronic signature, stored value, etc. • Does the authentication method leverage a device that is itself used for multiple purposes? e.g., PC, PDA, phone, etc.
Strategic Fit (corporate/ system)	Relative Security	• How strong is the authentication? • How secure is the implementation? • Is it adequate for the information being protected? • Does it meet regulatory requirements (if any) for the protection of information?
	Interoperability/ Back-end Integration	• Does the authentication solution work natively with multiple products? • Does it work only with the installation of additional software? • How easy is it to integrate with back-end resources or applications? What resources and applications need to be supported?
	Robustness/Scale	• Does the authentication solution scale to the degree required now? • Three years from now?
	Future Flexibility	• What future options may be available from the selection of this authentication solution (whether you currently intend to use them or not)? • What future options might be of interest?

Authentication protocols

one-way
- password
- challenge/response
- public-key

two-way (mutual authentication)
- trusted intermediary (Kerberos)
- public-key

issues
- secrets on server
- vulnerabilities of protocol -- cleartext, replay, dictionary attack, other?
- performance of protocol
- cost (user/server)
- convenience

Athentication protocols

```
USER                                    HOST database

----- name, password -->                clear, crypt, hash
----- name, securid# -->            server software
```

Our ncp asnmt 3?

```
**USER's machine***
--------name  ------>                   clear/hashed
              <------ R  --------  challenge (nonce)
--------- (R)key ------>          (encrypt or keyed-hash)

-------name (time)key ----->            clear/hashed
```

Our ncp asnmt 4

```
-----------name ------>             (Novell 3) hash of key H(key)
          <------------- R ------- challenge
----- H(R,H(key)) ------>

---------name----->                KDC (clear) (more later, Kerberos)
           <------(ticket)key ----

-----name--->                      NT (hash of key), LAN mgr
              <------------R---  challenge
-----(R)H(key) ------->
```

Mutual authentication

```
**USER's machine***                 HOST

------name ---->                     clear/crypt
                    <--------R1 -------
------(R1)key -------->
--------R2 ----------->
                 <-------(R2)key ---------
```

- later: KDC (Kerberos) can provide mutual authentication
- authentication can be done with public/private keys too (later)

Authentication vulnerabilities

- eavesdropping (sniffers)
- password database
- replay (need random numbers, good time stamp, sequence number)
- offline guessing (dictionary attack, know R1 and $(R1)_{key}$)
- Network session maybe hijacked after authentication! ☹

Passwords -- what you know

- easy to remember
- hard to guess
- historical: military, espionage

password vulnerabilities
- easy to guess
- read file where passwords are stored
- eavesdropper/sniffer
- sys mgr may know/decrypt
- off-line search
- strong passwords: inconvenient, people write them down!

Strong passwords

- special characters, numbers, upper/lower case
- would like 64 bits of randomness, so:
  - with 6-bit alphabet would need 11 characters
  - pronouncable, need 16 characters
  - choose your own, need 32 characters
- human mind is not a good password storage device

"Yes, I use my dog's name as my password.
My dog's name is 4X(Im,q@V-2
and I change his name every 90 days."

Password storage

- clear text (secret file, security through obscurity -- Ha!)
- encrypted
- encrypted on authentication server (kerberos, securid server)
- hashed (one-way encrypted or hash function)
- hashed and shadowed

UNIX passwords

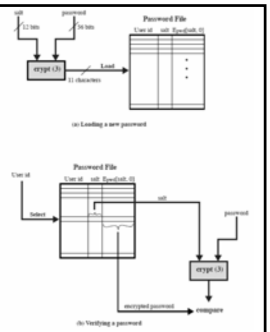
```
root:ab7er6hjkjase:0:0:root:/root:/bin/csh
bin:x:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/sbin:/sbin/nologin
adm:x:3:4:adm:/var/adm:/sbin/nologin
lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin
ftp:x:14:50:FTP User:/var/ftp:/sbin/nologin
nobody:x:99:99:Nobody:/:/sbin/nologin
ntp:x:38:14:ntp:/etc/ntp:/sbin/nologin
apache:x:48:48:Apache:/var/www:/bin/false
nscd:x:28:28:NSCD Daemon:/:/bin/false
ldap:x:55:55:LDAP User:/var/lib/ldap:/bin/false
xfs:x:43:43:X Font Server:/etc/X11/fs:/bin/false
gdm:x:42:42::/var/gdm:/sbin/nologin
dunigan:x:23673:20:TomDunigan:/home/dunigan:/bin/csh
```

- 8 characters (truncated)
- 2^{56} possibilities (7-bit ASCII)
- assigned by sys mgr
- change with passwd
- hashed and encoded in /etc/passwd
  - Hopefully shadowed
- portable hash string
- newer OS's provide MD5 encoding, longer phrases

Unix passwords

- user enters name and password
- host hashes password and compares
- 13 character string (Dqd2z5wN/3xyE)
- 2 character salt (random, 4096)

```
(void) time(&salt);
saltc[0] = salt & 077;
saltc[1] = (salt>>6) & 077;
... asciify
return(crypt(pwbuf,salt));
```



salt
- reduces identical password encodings
- discourages pre-hashed dictionary
- does not affect time to search for one password

crypt()
- uses 25 iterations of modified DES (EP is different, thwart DES hardware),
- encrypts 0 using salt/pwbuf as "key",
- encode 64-bit cipher output to 6-bit alphanumerics

Password vulnerabilities

- ask user
- guess
- brute force (1000 MIP years)
- dictionary attack (CRACK)
- eavesdrop (sniffer)
- shoulder surfing
- failed attempts not logged (WWW)
- trojan horse/library

Table 18.3 Observed Password Lengths [SPAF92a]

Length	Number	Fraction of Total
1	55	.004
2	87	.006
3	212	.02
4	449	.03
5	1260	.09
6	3035	.22
7	2917	.21
8	5772	.42
Total	13787	1.0

Table 18.4 Passwords Cracked from a Sample Set of 13,797 Accounts

Type of Password	Search Size	Number of Matches	Percentage of Passwords Matched
User/account name	130	368	2.7%
Character sequences	866	22	0.2%
Numbers	427	9	0.1%
Chinese	392	56	0.4%
Place names	628	82	0.6%
Common names	2239	548	4.0%
Female names	4280	161	1.2%
Male names	2866	140	1.0%
Uncommon names	4955	130	0.9%
Myths & legends	1246	66	0.5%
Shakespearean	473	11	0.1%
Sports terms	238	32	0.2%
Science fiction	691	59	0.4%
Movies and actors	99	12	0.1%
Cartoons	92	9	0.1%
Famous people	290	55	0.4%
Phrases and patterns	933	253	1.8%
Surnames	33	9	0.1%
Biology	58	1	0.0%
System dictionary	19683	1027	7.4%
Machine names	9018	132	1.0%
Mnemonics	14	2	0.0%
King James bible	7525	83	0.6%
Miscellaneous words	3212	54	0.4%
Yiddish words	56	0	0.0%
Asteroids	2407	19	0.1%
TOTAL	62727	3340	24.2%

CRACK -- p1Ssw0rd crack3r

- Dictionary and variations, guess and verify
- widely available (others: John the Ripper, plus crackers for Windows password databases)
- can use multiple machines (parallel)
- Tries variations of user names, gecos field then dictionary
- rule based ../Scripts/dicts.rules
- caps, substitution, reverse, add a number

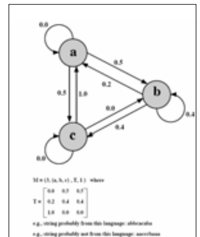
```
# Pluralise every significant one of the above
# MORE-THAN 2, NOT-CONTAIN ANY NON-ALPHA, LOWERCASE, PLURALISE
>2!?Alp

# Any alphaword >2 & <8 chars long, append a digit or simple punctuation
# since few ppl add non alpha chars to a already non-alpha word
# MORE-THAN 2, LESS-THAN 8, NOT ANY NON-ALPHA, LOWERCASE, APPEND <whatever>
>2<8!?Al0
>2<8!?Al1
>2<8!?Al2
>2<8!?Al3

# Lowercase every pure alphabetic word and reverse it
# MORE-THAN 2, NOT-CONTAIN ANY NON-ALPHA, LOWERCASE, REVERSE
>2!?Alr
```

Lots of password cracking software for various applications

countermeasures

- shadow file
- enforce strong passwords
  - Minimum length
  - combo of numerics, alphabetics, special characters
  - Language models (Markov →)
  - CRACK-like dictionary tests
    - Bloom filter Hash($word_i$)=y (y [0,N-1])
    - N-bit "hash" table, reject if $table_y$ is set
  - OS generate list
- scan for weak passwords (CRACK)
- password aging
- eliminate dormant accounts
- log failures (?)
- disable after n failures (?)
- trusted hosts (rlogin)
- ssh public/private key
- Kerberos

Don't use re-usable passwords!



Markov model
- language letter transition probabilities (digrams, trigrams)
- Reject if password fits model

One-time passwords

- SecurID or SafeWord tokens
- challenge/response (SecureNet, SNK)
- code-book
- Skey (OPIE)
- requires OS mods (login, ftpd, su) or special login shell/account
- may require server (Radius, securid/ACE) (and tokens … $$)
- new user setup harder

SecurID

- 2-factor authentication (card and PIN)
- need to issue cards (battery life?)… costs $$
- need ACE server (DB of username + card secret)
- need to modify login/su on enterprise servers to use ACE server
- hash of time and card secret (64-bit), every 60s
- login asks for name and current hash and PIN
- Brainard's proprietary hash reverse engineered
- security in card secret, not in secrecy of hash algorithm

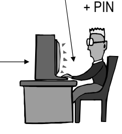
+ PIN

ACE server

Given userid, PIN, hash:
- get token secret from DB (userid)
- read clock and hash (time + secret)
- if "close" (try range of times), then OK

modified login

host



skey/opie

- challenge/response
- public domain (Skey, OPIE) and commercial clients for MAC/PC
- use from MAC/PC/workstation
- need password list for Xterminal or vt100 (or use PDA)
- based on Lamport paper and a one-way function (hash)
- modify (PAM) login/ftp/su etc.
- can configure to allow only skey logins
- can restrict user logins (net,host,tty)
- can use UNIX password from console

Skey implementation

- client needs *key* program
- Server data files /etc/skeykeys /etc/skey.access
- server needs *keyinit* plus modified *login, ftpd, su*

```
#ifdef KEY
permit_passwd = keyaccess(pwd, tty, hostname, (char *) 0);
pp = key_getpass("Password:", pwd, permit_passwd);
p = key_crypt(pp, salt, pwd, permit_passwd);
#else /* KEY */
pp = getpass("Password:");
p = crypt(pp, salt);
#endif /* KEY */
```

- with *keyinit* user creates hash using a seed, count, and passphrase
- H(H(H(.......H(seed,phrase))))...) count hashes
- host stores username,count,seed,hash

```
/etc/skeykeys
lpz 0037 ms68016          5f7fe2ac49089496  Sep 04,1996 16:15:38
phil 0032 ms08157          6d516f9931c703d6  Jul 16,1996 23:34:55
jgreen 0097 ra57824          a395980a0af44403  Apr 18,1996 13:48:01
mii 9999 ms34539          be21074bfa31b842  Feb 05,1996 15:31:36
```


Using skey

need client with *key* command

- user sends name
- host sends challenge (count-1, seed)
- user calculates $H_{count-1}$ (seed,passphrase) and sends encoding of hash, z
- hosts does one more hash, H(z), and compares to database
- if ok, user is logged in
- host stores count-1, and hash, z, from user
- telnet thdsun.epm.ornl.gov

```
login: dunigan
s/key 87 ic69188
(s/key required)
Password:
```

(in another window or with skey app.)

```
key 87 ic69188
Reminder - Do not use this program while logged in via
  telnet or rlogin.
Enter secret password:xxxxxxxxxxxxxx
GARB OVAL FIB AGEE BEAN AMES
```


Authentication summary

- two-factor authentication best
- use strong passwords, but still no help against sniffing!
- avoid re-usable passwords for login
- even with skey/securid -- session can be hijacked after authentication! ☹
- encryption (ssh) can prevent sniffing and hijacking
- strong passwords can thwart dictionary attacks
- long pass phrases thwart brute force (ssh, opie, or PGP)
- strong passwords are needed for protecting private keys
  - in public key crypto (netscape, pgp, ssh) -- private key file is encrypted with 3DES/IDEA/etc.

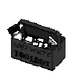
Hash functions

MD5, SHA, RIPEMD-160, tiger, panama, whirlpool
fun things you can do with hash functions

PAIN privacy, authenticity, integrity, non-repudiation

Crypto Toolkit

1. secret-key crypto
2. public-key crypto
3. big-number math
4. random numbers
5. prime numbers
6. hash functions



Hash functions

one-way function:
transformation of a message of arbitrary length into
a fixed-length (128+ bit) number (hash, checksum,digest,fingerprint)

- uses
  - message/file integrity
  - message authenticity (MAC)
  - detect change (error/malicious)
  - password hashing
  - in digital signatures (motivation)
  - key fingerprints
  - authentication (skey, SecurID)
  - pseudo-random numbers
  - encrypting (exportable)

hashing zoo
MD2,MD4,MD5
SHA
RIPEM
PANAMA
WHIRLPOOL
CBC encryption
Tiger
……

TOO simple hashes

- parity bit
- XOR of blocks or chars
- checksum (UNIX *sum* 16-bit)
  sum file
- TCP/IP/UDP checksums
- CRC (division instead of addition)
  - Remainder of division by polynomial
  - CRC32 fast: shift, XOR, table lookup
  - detects more bit error patterns
  - used for error detection (Ethernet)
  - more later on poly arithmetic

all easily forged
active adversary vs just error detection

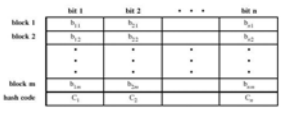


Figure 11.7 Simple Hash Function Using Bitwise XOR

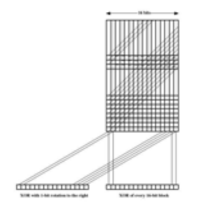


Figure 11.8 Two Simple Hash Functions

Strong hash

- z=H(m) is easy/fast to compute
- given z, computationally infeasible to find m such that
  H(m)=z (one-way)
- given m, infeasible to find an m' such that
  H(m')=H(m) (weak collision resistance)
- infeasible to find two random messages m and m' such that
  H(m') = H(m) (strong collision resistance), birthday attack
- strength in proportion to size of z (at least 160 bits)
- could choose any compression function (strength vs speed)
- no unconditionally provable one-way functions are known
- can only prove a function fails ☹
- the test of time

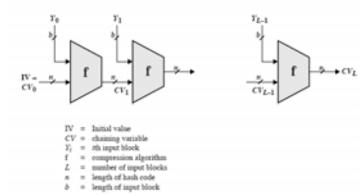
Note: there will be collisions, since mapping lots of bits into a few bits.

Hash algorithms

- recursive block, with padding and length, initialization vector (IV)
- multiple rounds/steps
- compression function
  - logical functions, shifts, additive constants
  - Or based on encryption function (whirlpool)
- 32-bit (efficient,byte-order?)
- avalanche (bit propagation)
  - one-bit change in message, results in massive change of hash value
- hash value "random"

Add in some text, some constants (πϕ√3), and stir

Structure of hash functions



- Input broken into blocks, last block padded with length value
- Each block is passed into compression function along with output from previous step
  compression function is usually made up of multiple rounds of "mixing"
- Output of final step is hash
- First step uses a (fixed) initialization value (IV)

Blocking, padding, and hashing

- Pad message with length info out to multiple of block size
- Process a "block" at a time (512 bits MD5, SHA-1; 1024 SHA-512)
- Iterate accumulating "running hash"

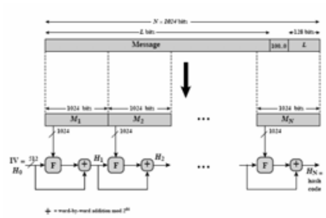


Figure 12.1 Message Digest Generation Using SHA-512

MD5

- son of MD2, MD4 (find collisions in a minute)
- 128-bit digest
- 32-bit word units, 512-bit blocks
- 4 rounds of 16 steps per block
  one word per step
- used by PGP, IP security, Tripwire, Skey, software dn.
- Weak? collision found with a specific IV…worse
- IV: variations on 0123456789abcdef
- 64 additive constants T_i derived from sin(i)
- 32-bit addition (e.g., mod 2^{32})
- Four round functions (32-bit words)
- Pad message with 10000…0 and 64-bit length so last block is 512 bits

MD2
256-bit blocks
128-bit hash
16 rounds
XOR with bytes of pi

MD4
512-bit blocks, 128b hash
3 rounds of 16 steps
3 primitive functions
Steps are not chained

Round	Primitive function
1	$F(b,c,d) = (b \wedge c) \vee (b' \wedge d)$
2	$G(b,c,d) = (b \wedge c) \vee (b \wedge d')$
3	$H(b,c,d) = b \oplus c \oplus d$
4	$I(b,c,d) = c \oplus (b \vee d')$

MD5 round

- Round processes 16 words (512 bits)
- Round functions F,G,H,I
- Constants T[i]
- X[i] i^{th} word of block
- 16 words are permuted in rounds 2, 3, and 4
  - (1 + 5i) mod 16
  - (5 + 3i) mod 16
  - 7i mod 16
- Feed 4x32-bits words to next round or result
- Start with 4x32-bit IV

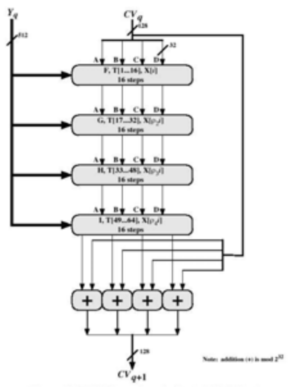


Figure 12.2 MD5 Processing of a Single 512-bit Block

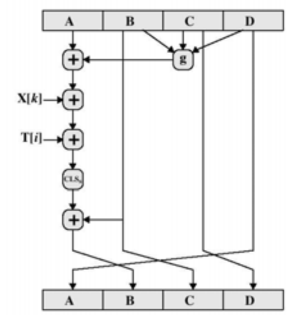


Figure 12.3 Elementary MD5 Operation (single step)

32-bit unit (A,B,C,D,X_k,T_i)

32-bit addition

Primitive functions (f,g,h,i)

Data word (X_k)

Additive constants (T_i)

Circular left shift (rotate word)

Input words are shifted each step

SHA

- 160-bit digest developed by NIST
- 32-bit word units, 512-bit blocks
- 4 rounds x 20 steps per block
- NIST design choices undocumented?
  - similar to MD4
- slower, stronger than MD5
- IV: variations on 0123456789abcdef
- 4 additive constants K_t SQRT{2,3,5,10}
- Rotates (S) and addition mod 2^{32}
- Primitive function

Step
0-19 $F(b,c,d) = (b \wedge c) \vee (b' \wedge d)$
20-39 $F(b,c,d) = b \oplus c \oplus d$
40-59 $F(b,c,d) = (b \wedge c) \vee (b \wedge d) \vee (c \wedge d)$
60-79 $F(b,c,d) = b \oplus c \oplus d$

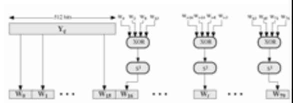


Figure 12.7 Creation of 80-word Input Sequence for SHA-1 Processing of Single Block

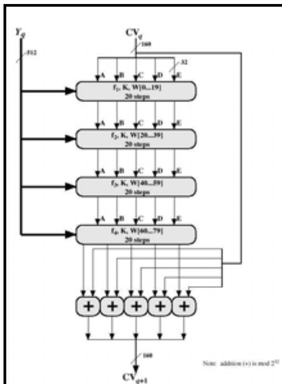


Figure 12.5 SHA-1 Processing of a Single 512-bit Block (SHA-1 Compression Function)

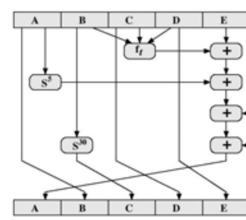


Figure 12.6 Elementary SHA Operation (single step)

Input (W_t) words XOR with 1-bit rotate of previous input words. Constants K_t

SHA variations

Table 12.3 Comparison of SHA Properties

	SHA-1	SHA-256	SHA-384	SHA-512
Message digest size	160	256	384	512
Message size	$< 2^{64}$	$< 2^{64}$	$< 2^{128}$	$< 2^{128}$
Block size	512	512	1024	1024
Word size	32	32	64	64
Number of steps	80	80	80	80
Security	80	128	192	256

Notes: 1. All sizes are measured in bits.
2. Security refers to the fact that a birthday attack on a message digest of size n produces a collision with a workfactor of approximately $2^{n/2}$.

MD5 vs SHA

- SHA more secure (160 vs 128), **neither are adequate today** – use 256 or 512
- MD5 vulnerable to cryptanalytic attack
- $10M ('94 dollars) find a collision in 24 days
- MD5 faster (see table →)
- Both are simple
- Big vs little endian

```
                     MD5   SHA
32-bit adds           4     4
logical              2-3   2-4  varies per step
rotates               1     2
----------------
total CPU/round       8   8-10

mem reads             2     2
reg reads             4     5
reg writes            1     2
---------------
total mem/round       7     9

total rounds         64    80
```

RIPEMD-160

- avoid MD4/5 weaknesses – designed to resist cryptanalysis
- 160 bit output (vs 128)
- 10 rounds (2x5) of 16 steps
- steps like MD5 plus a rotation
- parallel steps -- stronger against collisions
- word permutations to increase separation
- strong circular left shifts (shifts of 5 to 15 bits)
- 32-bit addition
- Round constants (SQRT and cube root (2,3,5,7))
- 5 primitive functions
- slightly slower than SHA or MD5

Step	
0 -15	$F(b,c,d) = b \oplus c \oplus d$
16-31	$F(b,c,d) = (b \wedge c) \vee (b' \wedge d)$
32-47	$F(b,c,d) = (b \wedge c') \oplus d$
48-63	$F(b,c,d) = (b \wedge d) \vee (c \wedge d')$
64-79	$F(b,c,d) = b \oplus (c \vee d')$

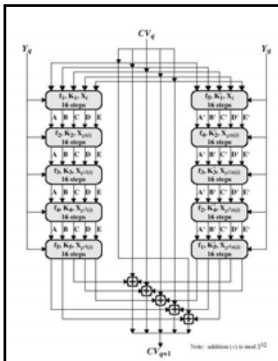


Figure 12.8 RIPEMD-160 Processing of a Single 512-bit Block (RIPEMD-160 Compression Function)

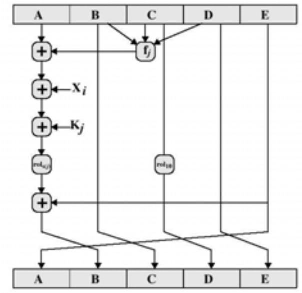


Figure 12.9 Elementary RIPEMD-160 Operation (single step)

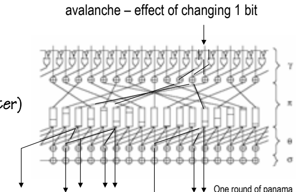
16-words (X) are permuted for each round. Constants K_j

Table 12.8 A Comparison of MD5, SHA-1, and RIPEMD-160

	MD5	SHA-1	RIPEMD-160
Digest length	128 bits	160 bits	160 bits
Basic unit of processing	512 bits	512 bits	512 bits
Number of steps	64 (4 rounds of 16)	80 (4 rounds of 20)	160 (5 paired rounds of 16)
Maximum message size	∞	$2^{64} - 1$ bits	∞
Primitive logical functions	4	4	5
Additive constants used	64	4	9
Endianness	Little-endian	Big-endian	Little-endian

panama

avalanche – effect of changing 1 bit



8 bits affected in 1 round

- 32-bit word based
- XOR, rotates,OR,permutes
- like linear feedback shift register (later)
- 256-bit input blocks
- IV: 0
- 32 rounds, 4 steps/round
- strong: 1KB chaining state, 256-bit output
  - MD5/RSA/RIPEM have only 128 or 160 bit state
- nonlinear, fast diffusion (avalanche: 4 rounds)
- faster than MD*/SHA/RIPEM for large blocks
  - Slow for small blocks, have to prime and drain state

tiger

- 64-bit word based
- XOR, add, subtract, multiply (5,7,9)
- permute, shift, 4 S-boxes (8x64) (substitution)
- 3 rounds, 8 steps/round
- nonlinearity from S-boxes
- strong avalanche: 3 steps
- 512-bit input blocks
- 192-bit output (3x64)
- IV: variations on 0123456789abcdef
- fast on 64-bit CPUs

whirlpool

- Uses AES-like encryption function (W) to mix bits
  - Based on polynomial arithmetic but fast (shift's and XOR's)
  - Added to OpenSSL 0.9.9
- 512-bit hash
- More secure? … test of time

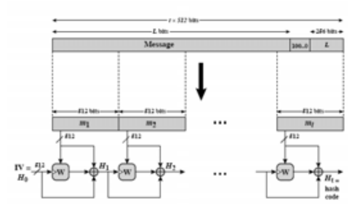


Figure 12.6 Message Digest Generation Using Whirlpool

Things to do with a hash

- file checksums (tripwire/software distribution)
- user authentication, one-time password (skey, Securid)
- digital signatures
- message authentication MAC (keyed hash)
- encryption
- pseudo random number generation for keys, primes, nonce…
- mixing function for hardware random bits
- key update with master key K, H(K,r_i),
  r_i is known random value
- distill passphrase to an encryption key (PKCS5)

File integrity

- shareware file signatures
  - –Create a separate signature file with pgp –sab file.txt
    md5sum mydist.tar > mydist.md5
- vendor patch files
- checksums of your/system files (e.g., tripwire)
  - –Protect the checksum file (offline, burn a CD, digitally sign)

```
    From gnupg.org download page (integrity check) MD5
b1890f5dfacd2ba7ab15448c5ff08a4e  gnupg-1.2.6.tar.bz2
56b10a6f444fff2565f4d960a11b2206  gnupg-1.2.6.tar.gz
3d5199fd729e2cf254a267c6935eeeaf  gnupg-1.2.5-1.2.6.diff.gz
```

Hacker could modify tar file and fix MD5 checksum on website, so digital signatures of tar files are even better.

```
gpg --verify gnupg-1.2.6.tar.bz2.sig
```


Hashing -- tripwire

- Verify file integrity (intrusion detection, forensics)
- Files: policy, database(hashes), reports, keys
  - –Policy says which files to check and their importance
- Hashes: CRC32, MD5, SHA, HAVAL
  "if you can't tie a good knot, tie lots of them"
- Hash DB and policy file protected with ElGamal pub/private keys (or read-only media)
- Create initial file signatures ("clean" system)
- Verify: reports changes/additions, notes critical files
- Troubles with legitimate file changes – data base update

Can tripwire detect a trojan tripwire ????
Are YOU running tripwire?

Authenticating with a hash

- Opie/skey one-time password
  H(H(H(… H(salt,password)))…)
- proving you know a secret without revealing the secret
  challenge-response (challenges R1 and R2 are random numbers)

```
USER workstation           HOST
name, R1 ---------->
                          <----- Hash(key,R1)-----
                           <------ R2 ------------
Verify Hash(key,R1)
------ Hash(key,R2)--->    verify
```

- digital signatures encrypt hash of message with sender's private key (compact, faster than public key encrypt of whole message)

Generating a key

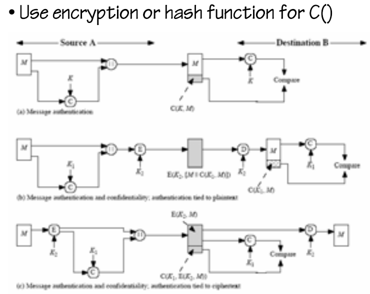
- Use hash function to generate one or more keys from a password
- key = hash(password)
- better: key = hash(password, salt, counter)
- or other variations see PKCS5 (RFC 2898)

MAC

Message authentication code (keyed hash)

- message integrity/authenticity (not privacy) MAC = C(K,M)
- protect against active attack (Eve can modify data stream)
- Use encryption or hash function for C()

Seal of authenticity
-from the king
-tamper-resistant



ssh v1 (weak C)

If MAC fails, save decryption time

Hash based MACs (HMAC)

Message authentication code (keyed hash)

- Try:
  - –send msg,Hash(msg) --nope
  - –send msg,Hash(key,msg) -- nope (appendable)
  - –send msg,Hash(msg,key) -- better
  - –send msg and only half of Hash(msg,key)
  - –send msg, Hash(key||opad,Hash(key||ipad,msg))
- HMAC (RFC2104) used in IPsec (truncate to 96 bits)
- attacking HMAC-MD5 much harder than attacking MD5
- OLDER ALTERNATIVE: use final output of CBC encryption as MAC or encrypt all of message including hash (ssh v1 used CRC32), or CMAC, but export-controlled ☹

HMAC

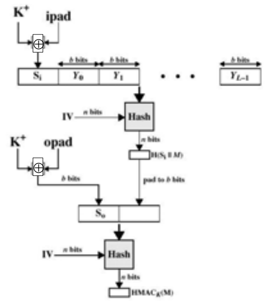


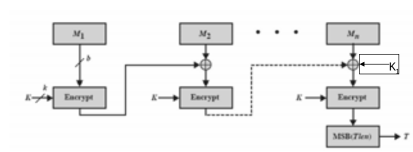
Figure 12.10 HMAC Structure

HMAC – keyed hash

- RFC2104 actually suggests using truncated final value, e.g. HMAC-MD5-96 or HMAC-SHA-80 used in IPsec and lots of network crypto
- Truncation (send only 96 bits of hash)
  - –Shorter message (faster transmission)
  - –Makes it harder for Eve to guess key
  - –But is hash value TOO short ? Can Eve find collisions (birthday attack)? Not really, can't do offline guessing without key – so you need to capture lots of M, $HMAC_k(M)$ pairs
- ipad and opad each flip half of the bits of the key and when each are then hashed, we generate 2 pseudorandom keys

MAC using encryption

- Use final output of chained encryption (CBC-MAC) FIPS 113 ANSI X9.17
- Use weak/faster checksum (CRC) but encrypt it (ssh v1)
- But encryption is often slower than HMAC and export-contolled ☹



- CBC-MAC has a block-extension attack, use CMAC instead
  - –Derive second key (K_1) from original key K and XOR in in final step

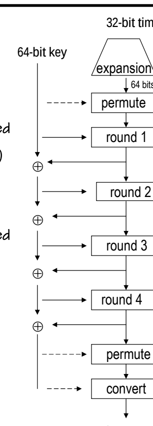
Universal MAC (UMAC)

- Combo of fast, simple hash then encrypt or HMAC for security
- Work with word-size units for fast computation
- Assume small (< 4Kbytes) messages (e.g. network) M= $M_1 M_2$ …
- Recipe:
  - –Generate 1024 32-bit subkeys K_i from key and a 512-bit hash key H
  - –HM = $(M_1+K_1)\times(M_2+K_2) + (M_3+K_3)\times(M_4+K_4) + \ldots$
    - Word-size dot product– fast (MMX)
  - –UMAC = HMAC-SHA1$_H$(HM || counter)
  - –Sender/receiver share key K and 64-bit counter, increment counter for each message
- IETF is looking to standardize (RFC 4418)
- Order of magnitude faster than HMAC
- Secure ? Test of time?

SecurID keyed hash

- •Each token has unique 64-bit key and a clock
- •Every 60 seconds key is hashed with time and display updated
- •4 rounds, 64 steps per round (shift, XOR, byte subtraction)
- •Key-dependent permutations
- •Round keys are XOR of key with output of previous round
- •The ACE server has token secret key DB and hash implemented in server to verify (userid,PIN,hash) from user/token (ref. securid.c)

Newer tokens use SHA?



Encyption with a hash function

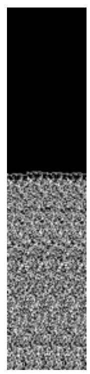
- compute a (pseudo) one-time pad with secret key
  - b_1= Hash(key, IV)
  - b_i = Hash(key,b_{i-1})
- XOR msg p_i with b_i $c_i = p_i \oplus b_i$
- receiver generates b_i and decrypts $c_i \oplus b_i \rightarrow p_i \oplus b_i \oplus b_i = p_i$
- stream cipher (more later)
- exportable
- used by RADIUS/TACACS+

Hash attacks

- clearly there are collisions, but it is infeasible to find one when you need it
- forgery -- find x' such that H(x') = H(x), weak collision
- find a pair x and x' such that H(x') = H(x), have Bob sign H(x) but then substitute message x'
  if 2^n hashes, birthday attack need try only $2^{n/2}$
- 2^{128} weak -- longer hash is better, use RIPEM/SHA (> 160)
- strength of hash is strength of compression function
- one-way: H(x) reveals nothing about x
- for a MAC if you can guess the key, then you can forge a message (dictionary attacks)
- Hashes used for random numbers (e.g., keys) need to withstand cryptanalytic attacks

MD5 (SHA-1?) collisions

- Code exists now to create collisions!
  - Examples, win32 .exe's with same 16-bit checksum, same 32-bit CRC and same MD5 hash!
  - Examples, 2 postscript files
  - Examples, public key cert that uses MD5
- MD5 signature on 2 files
  - Right, 2 files eventually differ (avalanche)
  - Left, 2 files collide, but end up with same hash
  - Collision software has to search a while to find right stuff to add to 2nd file
  - Takes advantage of length extension property of hash algorithms
- MD5 and SHA-1 are weak (HMAC's OK)



performance

- HMAC MD5 part of IPv6/IPsec specs
  - concern it is too slow, weak?
  - byte-order
  - slow: bit operations, carry-based scrambling, rotates
  - limited parallelism because of chaining
  - faster: PANAMA, Tiger, UMAC, Whirlpool
- XOR MAC (Bellare)
  - parallelizable, incremental (random block updates), provable
  - Hash_{key}(blockindex,msgblock)
  - XOR the hashes of each block with the hash of the counter, C
  - C = C+1; $z= H_k(C) \oplus H_k(1,M_1) \ldots \oplus H_k(n,M_n)$
  - send the message, hash, and counter – {M,z,C}
  - Receiver verifies using shared secret, k
- Maybe you don't want parallelism -- defeat high-speed attacks?

Hashing speed

```
openssl speed md5 sha1 rmd160
type             16 bytes     64 bytes    256 bytes   1024 bytes   8192 bytes
md5              18056.45k    63987.05k   189244.42k   372775.59k   513728.51k
sha1             16069.07k    53976.28k   137615.27k   231720.96k   289390.59k
rmd160           15465.35k    45976.60k   103148.46k   150882.65k   174637.06k
```

- Crypto++ benchmarks (hashing 1 MB)

```
Algorithm     data rate (MB/s)
MD5              217
SHA1              68
SHA-512           11
RIPEMD-160        53
Tiger             38
Panama           303
Whirlpool         12
```


Hashing software – command line

```
md5 file  (or md5sum)
sha file
```

OpenSSL supports

md2 md4 md5 sha sha1 sha256 sha512 rmd160

```
openssl md5 tst.c
MD5(tst.c)= 701e3948596ca492746863bff0288b7c
```

Hasing software – API


```
// compile with -lssl
#include <openssl/md5.h>

struct MD5_CTX mdctx;
unsigned char md5_hash[16];

MD5_Init(&mdctx);
MD5_Update(&mdctx,buffer1,lth1);
MD5_Update(&mdctx,buffer2,lth2);
MD5_Final(md5_hash, &mdctx);
  (also a  MD5(data,data_lth,hash)  )
```

Java

```
MessageDigest md = MessageDigest.getInstance("SHA");
md.update(Buffer1);
md.update(Buffer2);
byte [] hash = md.digest();
```


Hashing with EVP (EnVeloPe) in OpenSSL

```
#include <openssl/evp.h>
  const EVP_MD *m;
  EVP_MD_CTX ctx;
  unsigned char *ret;

  OpenSSL_add_all_digests ();
  if (!(m = EVP_get_digestbyname (“sha1”)))
    exit(1);
  if (!(ret = (unsigned char *) malloc (EVP_MAX_MD_SIZE)))
    exit(2);
  EVP_DigestInit (&ctx, m);
  EVP_DigestUpdate (&ctx, buf, len);
  EVP_DigestFinal (&ctx, ret, olen);
```

- Be crypto agile… in case SHA-nnn is found weak
  - Use config or protocol negotiation to select algorithm

HMAC programming

- Message integrity with keyed hash
- OpenSSL
  - Incremental HMAC_Init(), HMAC_Update, HMAC_Final
  - Single-shot

  HMAC(EVP_MD *evp_md , *key,keylth,*msg,msglth,*result, *resultlth)

```
unsigned char result[EVP_MAX_MD_SIZE];
HMAC(EVP_sha1(),hmackey,strlen(hmackey),msg, msglth, result ,&dlen);
```

- Procedure:
  - zero hmac field in message and do hmac, copy result to hmac field
  - To verify, save hmac from message, zero hmac field, do hmac and compare result to saved hmac from message
- Best practice: hmac key is different from encryption key

Next time …

Random numbers, steganography, and classical crypto

Assignment 3 (PGP) due Saturday (*make your directory and .plan word readable*)

Assignment 4 will take some debugging time … try it before next class.

Try to solve the two challenges on the class4 web page! ☺

class4

Lectures
1. Risk, viruses
2. UNIX vulnerabilities
3. Authentication & hashing
4. Random #s classical crypto
5. Block ciphers DES, RC5
6. AES, stream ciphers RC4, LFSR
7. **MIDTERM** ☹
8. Public key crypto RSA, D-H
9. ECC, PKCS, ssh/pgp
10. PKI, SSL
11. Network vulnerabilities
12. Network defenses, IDS, firewalls
13. IPsec, VPN, Kerberos, secure OS
14. Secure coding, crypto APIs
15. review