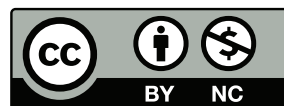

Cryptographic Engineering



Cryptographic Engineering — Issues

- Suppose we want to use crypto to protect files. Now what?
- What to encrypt?
- Where should keys be stored?
- What is the tradeoff between availability and confidentiality?

Why Encrypt Files?

- Theft of files
- Theft of backup media
- Theft of computer

Bad Reasons and Good

- Is there a flaw in the operating system's protection mechanisms?
Why can't the OS keep bad guys from the file?
- You don't trust the system administrator? Can the sysadmin steal the decryption key?
 - ☞ But — if you're using NFS, the file may reside on one (untrustworthy) machine, while the decryption is done on another
- Laptops have feet — a remarkably high percentage are stolen

Laptop Theft

September 17, 2000

IRVINE – Qualcomm founder Irwin Jacobs' laptop computer disappeared during a conference yesterday in an apparent theft that could put some of the company's most sensitive secrets at risk.

...

Jacobs said his laptop contained "everything," secret corporate information, including e-mail dating back years, financial statements and even personal mementos.

...

Though Jacobs' IBM ThinkPad PC is valued at about \$3,700, the value of the information it contained is incalculable to Qualcomm and to Jacobs.

Caveats

- Encrypting a file system provides confidentiality
- It generally does not provide integrity protection
- It may result in a *loss* of availability, if you lose the key

Encryption Options

- Manually encrypt/decrypt files
- Overlay encryption on top of the file system
- Encrypt an entire disk partition

Manual Encryption

- Very inconvenient to use
- Users are constantly supplying keys
- Most utilities won't have direct interfaces to the decryption function; you have to manually decrypt files before use
- Users *will* forget to re-encrypt files
- Important design principle: *make it easy for users to do the right thing*

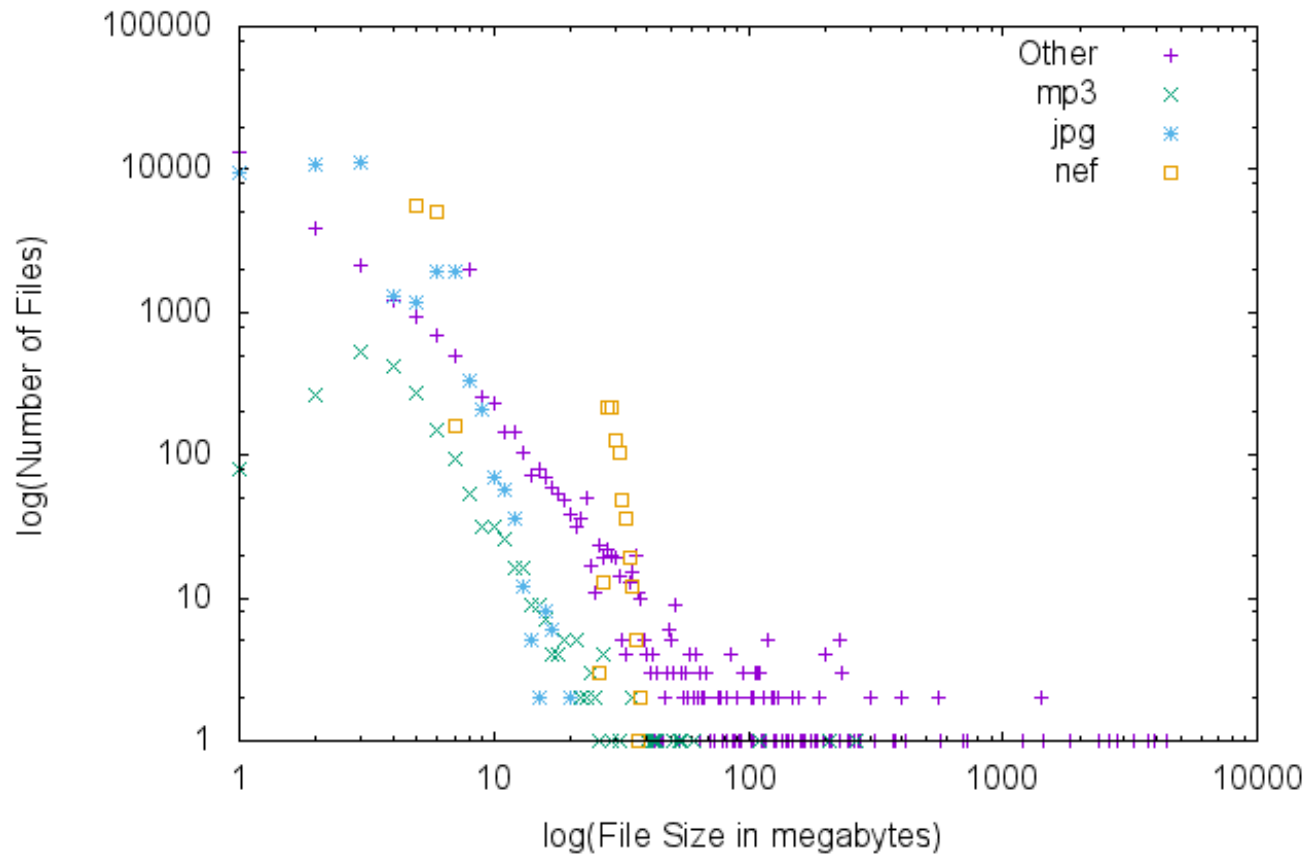
File System Encryption

- Some sort of overlay on real file system
- Encryption and decryption operate on individual files, but transparently to applications
- Directories are files, too, so filenames are encrypted

Problems With File System Encryption

- Metadata is not encrypted
- File lengths are not protected
- File name lengths are not well-protected

File Size Distribution



Population Count For a Few File Sizes

Type	1 MB	2 MB	3 MB	4 MB	5 MB	6 MB	7 MB	8 MB	28-32 MB
Other	13354	3874	2108	1205	923	682	498	1973	80
NEF					5629	5122	158		711
JPG	9574	10799	11176	1283	1190	1900	1945	336	
MP3	80	264	532	418	273	149	93	53	2

Files of 5 MB, 6 MB, or 28-32 MB are—on my disk—very likely to be NEFs; files under 5 MB are never NEFs. Files of 2–7 MB (except for 4 MB) are probably JPG or NEF; files 8 MB and larger are almost never JPG.

Encryption Using CFS

```
$ cattach /usr/mab/secrets matt
```

```
Key:
```

```
$ ls -ld /crypt/matt
```

```
drwx----- 2 mab 512 Apr 1 15:56 matt
```

```
$ echo "murder" > /crypt/matt/crimes
```

```
$ ls -l /crypt/matt
```

```
total 1
```

```
-rw-rw-r-- 1 mab 7 Apr 1 15:57 crimes
```

```
$ cat /crypt/matt/crimes
```

```
murder
```

```
$ ls -l /usr/mab/secrets
```

```
total 1
```

```
-rw-rw-r-- 1 mab 15 Apr 1 15:57 8b06e85b87091124
```

```
$ cat -v /usr/mab/secrets/8b06e85b87091124
```

```
M-Z,k\x{02C6}]\x{02C6}B\x{02C6}VM-VM-6A\x{02DC}uM-LM-__M-DM-\x
```

Doing the Encryption

- What mode of operation do you use?
 - ☞ CBC is a good choice
- Where does the IV come from? (Note: on Unix, must support seeks to any byte)
- Partial solution: encrypt each block separately; use block number as part of IV
- Must use some metafile for the rest of the IV. Solution must survive file copies, dump/restore, etc. (CFS uses .pvect files.)
- What about never-written blocks? On Unix, these read as all 0s

Providing Keys for Encrypted File Systems

- File system encryption: can be supplied by user
- Can have fine-granularity keying, per sub-tree
- Disk Encryption: one key per encrypted partition. Shared?
- In either case, once the key is supplied, you rely on OS protection mechanisms
- Bottom line: file system or disk encryption is useful if the threat is compromise from outside the boundaries of the machine: physical theft, remote file system, backup media, etc.
- It is not useful for intra-machine threats; an enemy who can bypass access controls can steal the key or the plaintext
- Encryption is not a substitute for operating system access controls

Disk Encryption

- Encrypt an entire disk or disk partition
- Protects everything, even the free space
- 👉 Very important, given that “delete” operations do not delete the data
- Useful for protecting swap area
- But — free space in encrypted section is not available for plaintext use, and vice-versa
- Problematic for remote file system

Protecting a Key Database

- How does the (symmetric key) trusted party safeguard its database of keys?
- Encrypt it? Where does the decryption key come from?
- One answer: supplied by operator at reboot time
- Another answer: store on a separate file system, so that the key and the encrypted data won't be on the same backup medium
- Tradeoff: availability versus confidentiality and integrity
- Use secure crypto hardware to decrypt database?
- Who has what sort of access, and what are their powers?

How Does a User Store a Key?

- Store key on disk, encrypted
- Generally decrypted with passphrase
- Passphrases are weak, but they're a second layer, on top of OS file access controls

Use Indirection

- Generate a random key to encrypt the data (DEK—Data-Encrypting Key)
- Use the user-supplied key to encrypt the DEK
- 👉 Make changing the password fast
- (Effectively) erase the disk is also very quick

Secure Cryptographic Hardware

- Can be used for users or servers
- More than just key storage; perform actual cryptographic operations
- Enemy has *no* access to secret or private keys
- Friends have no access, either
- Modular exponentiation can be done much faster with dedicated hardware

Hardware Issues

- Hardware must resist physical attack
- Environmental sensors: detect attack and erase keys
- Example: surround with wire mesh of known resistance; break or short circuit is detected
- Example: temperature sensor, to detect attempt to freeze battery

Limitations of Cryptographic Hardware

- Tamper-*resistant*, not tamper-*proof*
- Again: who is your enemy, and what are your enemy's powers?
- (Remember the “crypto in the hands of the enemy” problem.)
- How does Alice talk to it securely? How do you ensure that an enemy doesn't talk to it instead?
- What is Alice's *intent*? How does the crypto box know?
- What if there are bugs in the cryptographic processor software?
(IBM's 4758 has a 486 inside. That can run complex programs. . .)

Summary of Key Management and Key Handling

- Sharing cryptographic keys is a delicate business
- Protecting keying material is crucial
- There are no great solutions for general-purpose systems, though proper hardware can prevent compromise (but not misuse) of long-term keys

Random Numbers

- Random numbers are vital for cryptography
- They're used for keys, nonces, primality testing, and more
- Where do they come from?

What is a Random Number?

- Must be *unpredictable*
- Must be drawn from a large-enough space
- Ordinary statistical-grade random numbers are not sufficient
- *Distribution* not an indication of randomness: loaded dice are still random!

Generating Random Numbers

Anyone who considers arithmetical methods of producing random digits is, of course, in a state of sin.

—John von Neumann, 1951

Sources of Random Numbers

- Dedicated hardware random number sources
- Random numbers lying around the system
- Software pseudo-random generator
- Combinations

Hardware Random Number Generators

- Radioactive decay
- Thermal noise
- Oscillator pairs
- Other chaotic processes

Radioactive Decay

- Timing of radioactive decay unpredictable even in theory — it's a quantum process
- Problem: low bit rate from rational quantities of radioactive material
- Problem: not many computers have Geiger counters or radioactive isotopes attached...
- See <http://www.fourmilab.ch/hotbits/hardware.html> and <http://www.fourmilab.ch/hotbits/hardware3.html> for a description of how to do it...

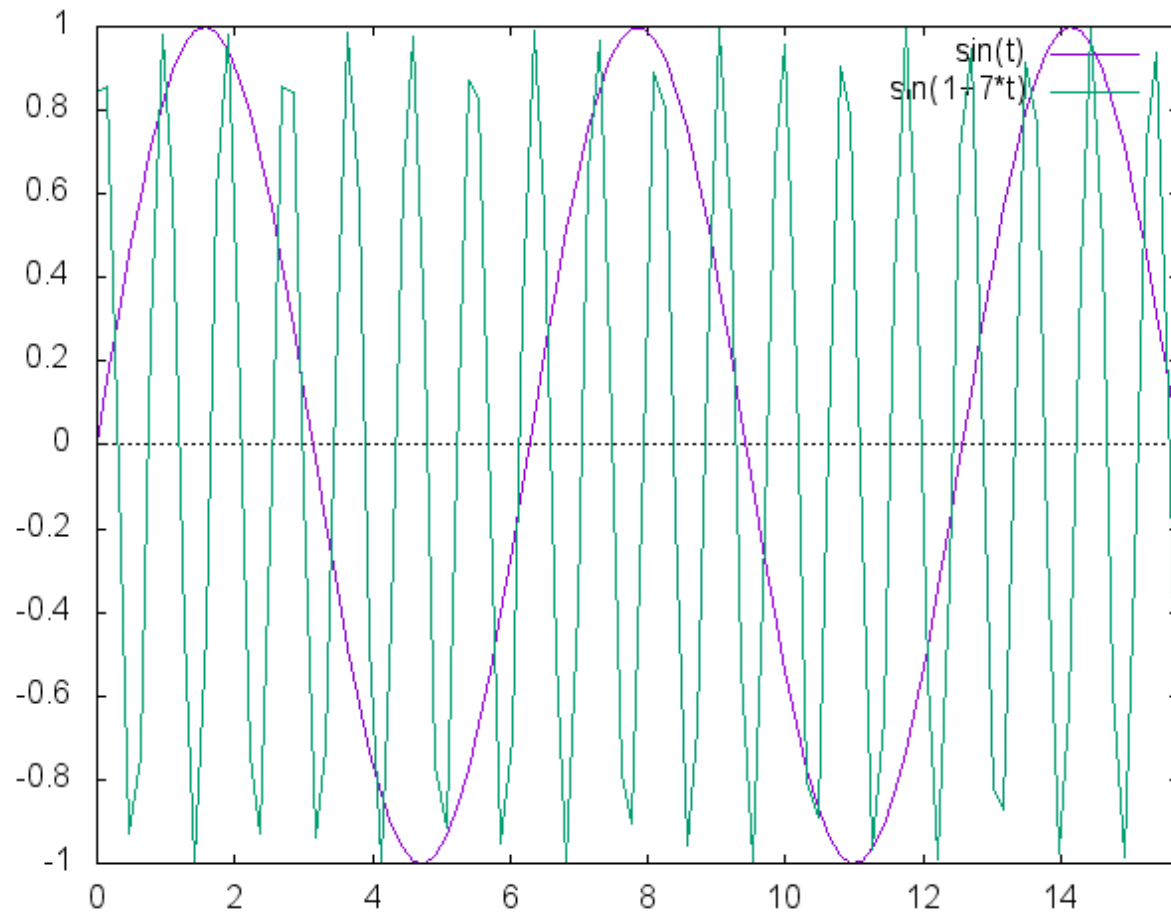
Thermal Noise

- Any electronic device has a certain amount of random noise (thermal noise in the components)
- Example: Take a sound card with no microphone and turn up the gain to maximum
- Or use a digital camera with the lens cap on
- Problem: modest bit rate

Oscillator Pairs

- Have a free-running fast R-C oscillator (don't use a crystal; you don't want it accurate or stable!)
- Have a second, much slower oscillator
- At each maximum of the slow oscillator, sample the value of the fast oscillator
- Caution: watch for correlations or couplings between the two

Dual Oscillator RNG



Other Chaotic Processes

- Mouse movements
- Keystroke timing (low-order bits)
- Network packet timing (low-order bits)
- Disk seek timing: air turbulence affects disk internals (but what about solid state disks?)
- ☞ At boot time, there's not much of this available
- Also: what if the enemy can observe the process?
- Cameras and Lava Lites[®]! (<http://www.lavarnd.org/>)

Problems

- Need deep understanding of underlying physical process
- Stuck bits
- Variable bit rate
- How do we measure their randomness?
- *Assurance* — how do we *know* it's working properly?

Software Generators

- Again, ordinary generators, such as C's `random()` function or Java's `Random` class are insufficient
- Can use cryptographic primitives — encryption algorithms or hash functions — instead
- But — where does the seed come from?

Typical Random Number Generator

```
unsigned int
nextrand()
{
    static unsigned int state = 1;

    state = f(state);
    return state;
}
```

What's wrong with this for cryptographic purposes?

Problems

- The seed is predictable
- There are too few possible seeds
- The output is the state variable; if you learn one value, you can predict all subsequent ones

A Better Version

```
unsigned int
nextrand()
{
    static unsigned int state;
    static int first = 1;

    if (first) {first = 0; state = truerand();}
    state = f(state);
    return sha1(state);
}
```

Much Better

- State is initialized from a true-random source
- Can't invert sha1() to find state from return value
- But there is a serious problem here. What is it?

State Space

- sha1() isn't invertible, but we can do a brute force analysis
- **state** is too short, and can be found in 2^{32} tries
- Estimated resources on a 3.4 Ghz Pentium: 3.6 hours CPU time; 150 GB
- Parallelizes nicely
- Need enough state — and hence enough true-random bits — that brute force is infeasible.

Private State

- An application can keep a file with a few hundred bytes of random numbers
- Generate some true-random bytes, mix with the file, and extract what you need
- Write the file back to disk — read-protected, of course — for next time
- What about stored VMs? Will they get the same seed each time?

OS Facilities

- Many operating systems can provide cryptographic-grade random numbers
- `/dev/random`: True random numbers, from hardware sources
- `/dev/urandom`: Software random number generator, seeded from hardware
- Windows has analagous facilities

A Well-Known Failure

- As noted, not much randomness is available at boot time
- But—that's often when key pairs are generated
- An RSA public key is the product of two “random” primes
- Might one be predictable?
- Heninger, Durumeric, Wustrow, and Halderman showed that many ssh keys have at least one predictable prime factor, for just this reason

DUAL_EC_DBRG: The NSA Back Door

- NIST decided to standardize a software PRNG

☞ This is a good thing

- NIST picked several designs—and the NSA persuaded NIST to include another based on elliptic curve cryptography
- It seemed odd—DUAL_EC is quite slow, since it's based on public key technology—but the NSA insisted that they needed it. They did need it, but not for the usual reason. . .
- At least one company, RSA, made it the default in their product, allegedly after being paid off

The Problem with DUAL_EC_DRBG

- The algorithm includes a “random” constant
- If it’s not random—if it’s the public key in an elliptic curve cryptosystem—anyone who can see enough of the output from the PRNG *and* knows the corresponding private key can predict all future output from the algorithm
- Many protocols do in fact transmit some random bits in the clear
- There have been public demonstrations that it’s exploitable under certain circumstances
- NIST has removed it from their standard, RSA has removed it from their code. . .

Hardware Versus Software Random Number Generators

- Hardware values can be true-random
- Output rate is rather slow
- Subject to environmental malfunctions, such as 60 Hz noise
- Software, if properly designed and written, is fast and reliable
- Combination of software generator with hardware seed is usually best

Summary

- To paraphrase Knuth, random numbers should not be generated by a random process
- In many systems, hardware and software, random number generation is a very weak link
- Use standard facilities when available; if not, pay attention to RFC 4086