Lessons Learned in Implementing and Deploying Crypto Software

Peter Gutmann

University of Auckland

What this Talk is About

"… the ways in which security-uneducated programmers can completely blow it on implementation of systems incorporating crypto…"

"… a 10 page rant about how people use crypto incorrectly…"

Intended Audience

Developers of security/crypto software

- Point out the biggest problem areas
- "Don't do that!"

Users of security/crypto software

- Things to watch out for

The Problem

Basic tools for strong crypto are widely available

- Anyone can obtain the code to do 3DES, AES, RSA, SHA-1, …

The snake oil crypto problem has been solved

- No it hasn't, it's just been moved elsewhere
- The determined programmer can produce snake oil using any crypto tools

But all of these problems are very well known!

- Obviously not, or they wouldn't occur so widely

Snake Oil Crypto

Obvious snake oil

"Our product uses a proprietary, patent-pending, military-strength, million-bit-key, one-time pad built from encrypted prime cycle wheels"

Non-obvious snake oil

"Our product uses Blowfish with a 448-bit key"

- Key is derived from an uppercase-only 8-character ASCII password

First-derivative snake oil: Naugahyde crypto

Naugahyde Crypto

Crypto software assumes a fairly high degree of crypto knowledge and motivation in users

- Users of the software are programmers, not cryptographers
- Motivation is "the boss said do it"
- If a security measure blocks functionality, disable the security measure

Result: Genuine Naugahyde crypto

Private Keys Aren't

PGP, circa 1991

> "never give your secret key to anyone else…always keep physical control of your secret key, and don't risk exposing it by storing it on a remote timesharing computer"

Typical usage, circa 2002

- Private keys are just another data item that can be copied around freely
- Re-using one key for everything is simple and convenient

Private Keys Aren't (cont)

Often motivated by certificate costs

- Company spends $495 and several hours' work creating a key and getting a Verisign certificate for it
- Most practical (in terms of time and money) application of this is to re-use it everywhere

Example: Company creates an encrypted file transfer system for a large customer

- 2048-bit PGP key used to encrypt files
  - Stored on disk in plaintext form, since the software runs as a batch process
- Encryption is ephemeral
  - Encrypted momentarily in transit, decrypted immediately on receipt

Private Keys Aren't (cont)

- Customer calls to say they've lost the private key, and can it be recreated for them
  - Developers decide how best to explain to the customer the meaning of the word "private" in "private key"
- A developer remembers that there's a copy stored with the source code on the developers' file server…
  - …and in other locations with the application binaries…
  - …and in SCCS…
  - …and on various developer machines…
  - …some of which were recycled for new employees with their contents intact
- The file server had recently had its drives upgraded
  - The old drives were sitting on a shelf in the server room

Private Keys Aren't (cont)

- The server was backed up regularly
  - Three staff members took it in turns to take the tapes home
  - Usual practice was to drop them in the back of the car until they were recycled
- It would have been almost impossible to destroy all the copies of this private key
  - This is a fine data backup strategy
  - Not so good for encryption keys
  - Correct recovery strategy: `pgp -kg`

Private Keys Aren't (cont)

CAs generate private keys for users and mail them out as PKCS #12 files

- Password is sent as separate mail or is easily guessed (8 characters, uppercase-only)
- This is standard practice for a number of CAs

CAs distribute their own private keys as PKCS #12 files

- The theory is that once installed, it makes the CA key trusted
- This "solution" is so common that it's warned about in the OpenSSL FAQ
- At least one computer security book contains step-by-step instructions on how to distribute your CA's private key to all users

Private Keys Aren't (cont)

Smart cards store private keys internally and don't reveal them

- "How can I use a smart card if I can't get at the key?"
- Attempted fixes are to…
  - Construct mechanisms for sharing cards across multiple machines
  - Generate the key externally and keep a copy after it's loaded onto the card
- Maybe the inconvenient fact that they keep private keys private is why crypto smart cards aren't taking off

Private Keys Aren't (cont)

If your product allows the export of private keys in plaintext form or plaintext-equivalent form (a widely-readable format), you should assume that your keys will end up in every other application on the system, and occasionally spread across other systems as well.

Everything is a Certificate

PKCS #12 files

- Quack like a certificate
- Contain the private key alongside the certificate
- Are often referred to as "digital IDs"
- Windows "Certificate Export Wizard" creates PKCS #12 files with private keys
  - Key can be exported without a password if the user just keeps clicking "Next"

Result: Users treat them like certificates, and not private keys

Everything is a Certificate (cont)

PGP

- Documentation clearly distinguishes public and private keys
  - ‘never give your secret key to anyone else…always keep physical control of your secret key, and don’t risk exposing it by storing it on a remote timesharing computer”
- Public and private keys are physically separated
  - – When exporting keys, PGP only exports public components…
  - – …even if the user explicitly forces the use of the private key file
- Safe-by-default key handling

Everything is a Certificate (cont)

Make very clear to users the difference between public and private keys, either in the documentation/user interface or, better, by physically separating the two.

Making Key Management Easy

The easiest key management technique uses fixed, shared keys

- Popular in sectors such as banking, who have a great deal of experience in handling confidential information

Also popular in other areas…

- EDI: Transmit the key in another message segment
- XML: Place it in a field tagged as <password> or <key>
- WEP

Making Key Management Easy (cont)

Public-key management made easy

- Everyone shares the same private key

Popular places to store private keys

- Encrypted file on your hard drive
- Smart card
- NFS share
- LDAP directory
- Web server

The last three are not isolated cases!

- "It's convenient"
- "What's the problem?"

Making Key Management Easy (cont)

Straight Diffie-Hellman requires no key management. This is always better than other no-key-management alternatives that users will create.

Making Key Management Easy (cont)

Example: STARTTLS

- Explosive growth in the last year or so
- Certificates used are
  - Self-signed
  - Signed by the default Snake Oil CA
  - Signed by unknown/test CA
  - Expired
  - Incorrect DN
- It's unauthenticated DH, but it's good enough

Making Key Management Easy (cont)

Unscientific claim:

> There is more mail being protected by STARTTLS than all other email encryption technology combined

Made possible by a combination of two things:

1. Out-of-the-box functioning in recent MTA releases
   - Postfix (easily enabled, English rather than M4)
     - Use the sendmail → Postfix upgrade to also move to STARTTLS
   - Some newer sendmail configs (e.g. Debian)
2. Use of unauthenticated DH

Result: Set it and forget it, it just works

Making Key Management Easy (cont)

How many people here:

- PGP-protect most of their email?
- S/MIME-protect most of their email?
- Have STARTTLS enabled in their MTA?
- Know they have STARTTLS enabled in their MTA?

All your Keys are Belong to Us

Key mismanagement is the single biggest problem in public crypto deployment

- Users don't understand the concept of public-key encryption (common)
  - Both sides share the key/all parties share the key
  - "The sender encrypts with the private key…or something"
- Users don't understand why "private key" has the word "private" in it (occasional)
- Sharing the private key is convenient (very common)
  - "We need to share the key across the Internet or *project goal* won't work"
  - "It cost us $ *xxx/yyy* hours' effort to get this key, we're not going through all that again"

All your Keys are Belong to Us (cont)

Much of the problem is social/financial

- Certificates are expensive to obtain
- Certificates are troublesome to obtain
- Users are given a considerable incentive to re-use certs/keys

Solution: Users have key-signing certs which they use to manage their own confidentiality keys

- Used by PGP
- Would deprive CAs of revenue → will never fly
- "We don't want to turn X.509 into PGP" — PKIX WG chair

All your Keys are Belong to Us (cont)

- Public-key encryption/signing is widely used in places where it's not needed
  - Integrity protection: Use a MAC (non public key-based message authentication code)
  - Establishing session keys for data encryption: Use DH key agreement
  - Just because your vendor tells you that you need a PKI doesn't mean you actually do

What Time is it Anyway?

Clocks on Windows (and other non-NTP) machines can be out by…

- Tens of minutes
  - Incorrect setting
  - Drift
- Hours
  - Incorrect setting
  - Daylight savings time adjust
- Days
  - Wrong time zone
- Weeks or months

What Time is it Anyway? (cont)

- Decades
  - Only noticed when the system rejected some certs with known validity periods
- Combinations of the above

Time zone issues

- Applications handle conversion differently
- Systems are misconfigured
- Impossible to compare some times, e.g. system in time zone A with DST vs. system in time zone B without DST

What Time is it Anyway? (cont)

Many security protocols assume perfectly synchronised clocks

- Extends the security baseline to something not normally regarded as security-relevant
- Almost all PKI mechanisms assume perfect, secure time management across all systems

How do you break a PKI protocol?

A. Factor the 2048-bit RSA modulus

B. Hypothesise and prove a protocol flaw using BAN logic

C. Change the system time

What Time is it Anyway? (cont)

Answer:

- C. Change the system time

System clocks are often deliberately set incorrectly to let expired certs/signed data be rejuvenated

- Shareware authors cottoned onto this one 15-20 years ago
- Palladium will save the day

Some protocols include unnecessary timestamps

- Don't broadcast the fact that your system clock is 30 minutes out
- cryptlib randomises 16 or even 32 bits of non-essential timestamps
  - No other app has ever complained

What Time is it Anyway? (cont)

Don't incorporate the system clock (or the other parties' system clocks) in your security baseline.

If you need synchronisation/freshness guarantees, use nonces (cookies).

What Time is it Anyway? (cont)

Use clocks as a means of measuring the passage of time rather than as an absolute indicator

- Client submits a nonce to a server
- Server replies with a signed/MAC'd reply indicating that the next update is in 15 minutes
  - Nonce guarantees freshness/prevents replays
  - Client can measure both message RTT and 15-minute interval time
- This works even if the client thinks it's September 1986

What Time is it Anyway? (cont)

In the presence of arbitrary end-user systems, relative time measures work. Absolute time measures don't.

RSA in CBC Mode

RSA encryption is always described as "encrypting with RSA"

- Non-cryptographers try to do exactly that
- Actual use is:
  - Encrypt a session key for a (fast) block/stream cipher with (slow) RSA
  - Encrypt the data with the fast block/stream cipher
- This is described as "RSA encryption" although the data is never encrypted with RSA

RSA in CBC Mode (cont)

Most toolkits don't allow bulk data encryption with RSA

JCE, in a fit of orthogonality, does allow this

- RSA in CBC mode with PKCS #5 padding
  - Break the data up into RSA-key-size chunks
  - Encrypt each chunk individually using RSA
- CBC mode and PKCS #5 are mechanisms for block ciphers, not public-key algorithms
- Why use a hammer to pound a screw when you have a wrench?

RSA in CBC Mode (cont)

Example: Programmer is asked to implement RSA encryption in Java

"No-one with an ounce of crypto knowledge would do it this way"

"It's exactly what you asked for"

- Customer threatens to withhold payment until it's fixed
- Programmer slips it in under "Misc.Exp." at five times the original cost

RSA in CBC Mode (cont)

Don't include insecure or illogical security mechanisms in your crypto tools.

Left as an Exercise for the User

Problems which developers couldn't solve are left as an exercise for the user

Entropy-gathering for crypto random number generators is typically done this way

- Netscape disabled BSAFE safety checks in order to allow the generator to run without proper initialisation

Left as an Exercise for the User (cont)

Simple safety check was added to OpenSSL 0.9.5 to test whether the generator had been properly initialised

An entry was added to the FAQ to explain this

- Later versions of the code were changed to display the URL for the FAQ

User responses…

- Seed the generator with a constant text string
- Seed it with DSA public components (which look random)
- Seed it with output from `rand()`
- Seed it with the executable image
- Seed it with /etc/passwd

Left as an Exercise for the User (cont)

User responses…

- Seed it with /var/syslog
- Seed it with a hash of files in the current directory
- Seed it with a dummy "random" data file
- Seed it with the string "0123456789ABCDEF"
- Seed it with output from the (uninitialised) generator
- Seed it with "string to make the random number generator think it has entropy"
- Downgrade to an older version of the generator that doesn't perform the check
- Patch the code to disable the check
- Later versions of the code added /dev/random support
  - Replace the /dev/random read with a read of a static disk file

Left as an Exercise for the User (cont)

Similar advice has been given on bypassing the JCE RNG

Based on user comments, quite a number of third-party apps had been using low-security cryptovariables from the uninitialised generator

Left as an Exercise for the User (cont)

If a security-related problem is difficult for a crypto developer to solve, there is no way a non-crypto user can be expected to solve it. Don't leave hard problems as an exercise for the user.

This Function can Never Fail

Software should never leak plaintext/crypto keys if a failure occurs

Example: Product used RSA encryption to wrap a session key

- Everything is tested and works fine
- The RSA key parameters were set up wrong and the RSA operation had failed
- The unencrypted session key was being sent over the wire
- At the other end, the RSA operation failed similarly, and the (untouched) session key was used to recover the encrypted data

This Function can Never Fail (cont)

Example: IIS returns data sent over an SSL connection back over the link as plaintext

- Race condition between two threads accessing the same buffer
- Thread 1 (receiver) decrypts data in the buffer
- Thread 2 (sender) sends data in the buffer over the network

This Function can Never Fail (cont)

Make security-critical functions fail obviously, even if the user ignores return codes.

Ensure red/black separation of sensitive and non-sensitive data.

Careful with that Axe, Eugene

Crypto protocol design is a fine art

- Most users who roll their own from some DES and RSA code get it wrong

Example: Use of DES in ECB mode

- Encrypts data in discrete 8-byte blocks
- Doesn't require initialisation vectors, cipher block chaining, etc etc
- Doesn't hide data patterns
- Vulnerable to trivial cut-and-paste attacks
- Warned against in every book on computer security

Careful with that Axe, Eugene (cont)

Example: Vendor builds VPN software using 3DES

- Encryption is done in ECB mode, which is easiest to use
- Leftover (1–7) bytes aren't encrypted
  - Consider what happens with interactive terminal traffic
- Product can be sold as "VPN with triple-DES encryption and 192-bit key"
  - You can even get this FIPS certified

Users are not crypto experts

- Users should not need to be crypto experts

Careful with that Axe, Eugene (cont)

Provide crypto functionality at the highest level possible in order to prevent users from injuring themselves and others through misuse of low-level crypto functions with properties they aren't aware of.

Further Work/Other Problems

What other problem areas exist?

How can we address the problem at the technical design level?

- Security target for crypto protocol design is provable security
  - "The weak points have nothing to do with mathematics [...] Beautiful pieces of mathematics were made irrelevant through bad programming" — Bruce Schneier
  - Looking at deployed security technology, you'd think the only (non-algorithmic) advances in crypto research in the last 10 years are HMAC and SPEKE
- Many users don't understand public-key encryption
- PKI is way too hard to use → users short-circuit security measures

Further Work/Other Problems (cont)

How can open source code prevent users from patching it to disable security measures?

- Sometimes proprietary software does have some advantages

Many key management issues are economic and social problems

- HCI issue
- Almost no work exists on making crypto usable

Recommended Reading

"Building Secure Software", John Viega and Gary McGraw (Unix-oriented)

"Writing Secure Software", Michael Howard and David LeBlanc (Windows-oriented)

- Get both of them, even if your primary work area is covered by the other book

"Network Security", Charlie Kaufman, Radia Perlman, and Mike Speciner

- Covers security solutions *and* practical problems/weaknesses

All three books have second editions coming out

Conclusion

Snake oil is rapidly becoming a thing of the past

- It's being replaced with naugahyde crypto

Problems are well-known…

…for very small values of "well -known"

Technical solution

- Follow simple guidelines to eliminate the major problems

Social/economic solution

- ?