Primitives

- Symmetric encryption algorithms
- Public key encryption
- Digital signatures
- Hash functions
- How do two parties agree on a session key?

Constraints

- Is there one party whom Alice and Bob both trust?
- How much do they trust this party?
- How much *should* they trust this party?
- Can they talk to this party before encrypting?
- Can they talk to each other before encrypting?
- Can they use public key cryptography?

Disclaimer

- We're not going to explore this in detail in this class
- This is a subtle and difficult field; even experts make mistakes
- We're going to concentrate on the engineering aspects

Needham-Schroeder

- Oldest cryptographic protocol in the open literature (Dec 1978)
- Symmetric and public key variants described
- Warning from the authors:

  > Finally, protocols such as those developed here are prone to extremely subtle errors that are unlikely to be detected in normal operation. The need for techniques to verify the correctness of such protocols is great, and we encourage those interested in such problems to consider this area.

They Were Right About the Difficulty

1981 Mistake found in symmetric variant (Denning and Sacco)

1994 Needham find a mistake in Denning and Sacco's fix

1996 New mistake found the public key variant of the original protocol

☞ All of these mistakes were blindlingly obvious once found. The public key variant, in modern notation, is only 3 messages!

Principles of the Symmetric Variant

- Each party should know the identity of the other
- Each party shares a secret key with a trusted, central authority
- This third party acts as an introducer: it vouches for Alice's identity, ensures that she's talking to Bob, and supplies a session key
- Random *nonces* used to avoid *replay attacks*
  - ☞ I send you a random number and expect you to return it to me, suitably protected

What's a Replay Attack?

- Enemy records your (encrypted) traffic
- At a suitable time, enemy retransmits the message
- It's encrypted, so the target thinks it's genuine
- Often used with cut-and-paste: combine pieces from different messages

Properties of Symmetric Needham-Schroeder

- Requires online trusted party
- Must be able to talk simultaneously to this party and to your correspondent — okay for some networks; lousy for secure phones
- What if that third party is untrustworthy or hacked?
- An enemy can read all past conversations, via access to the keys everyone shares with this party

The Public Key Variant

- The trusted party is only vouching for *identity*; it is not supplying keys
- If it is hacked, the bad guy can impersonate Alice or Bob, for future conversations, but can't read old ones
- Needham and Schroeder didn't know about certificates (invented by an MIT undergraduate in June 1978), so they spoke of caching
- With certificates, the trusted party can be *offline*
- Alice and Bob can send each other their certificates
- This greatly reduces its vulnerability to attack
- Nonces still used to prevent replay

Sending Secure Email

- Alice must have a key for Bob before sending the mail
- Bob may be offline; Alice either must have the key herself or ask a third party
- Use of certificates allows Alice to be offline, too
- Generally use timestamps for freshness
- Note: Alice and Bob must have roughly-synchronized clocks
- Bob must retain old emails within clock skew window
- Could use sequence numbers if email is reliable

Does Alice Sign the Plaintext or Ciphertext?

- If Alice signs the plaintext and encrypts the signed message, attackers can't see who sent the message
- If Alice encrypts the plaintext and signs the ciphertext, her mailer can verify the origin without access to Alice's private key
- Note: signing the ciphertext is somewhat stronger cryptographically

What Do I Mean “Access to Alice’s Key?

- Cryptographic keys are very sensitive, especially long-term keys
- Must be carefully guarded
- Protecting keys properly is a major challenge on today’s systems

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 Alice Store her 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

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?
- 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*?
- What if there are bugs in the cryptographic processor software? (IBM's 4758 has a 486 inside.)

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` for a hardware description

Thermal Noise

- Any electronic device has a certain amount of random noise
- 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 (`http://www.lavarnd.org/`)
- 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

Other Chaotic Processes

- Mouse movements
- Keystroke timing (low-order bits)
- Disk seek timing (air turbulence affects disk internals)
- Cameras and Lava Lites®!

Problems

- Need deep understanding of underlying physical process
- Stuck bits
- Variable bit rate
- How do we measure their randomness?

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 md5(state);
}
```

Much Better

- State is initialized from a true-random source
- Can't invert md5() to find state from return value
- Actually, we can: `state` is too short, and can can be found in 2^{32} tries

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

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

- Wagner and Goldberg attacked Netscape 1.1's cryptographic random number generator
- Generator was seeded from process ID, parent process ID, and time of day
- `ps` command gives PID and PPID
- Consult the clock for time of day in seconds
- Iterate over all possible microsecond values
- Note: they did this by reverse-engineering; they did not have browser source code
- `http://www.cs.berkeley.edu/~daw/papers/ddj-netscape.html`

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 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