**Properties of Output Feedback Mode**

- No error propagation
- Active attacker can make controlled changes to plaintext
- OFB is a form of stream cipher

**Properties of Counter Mode**

- Another form of stream cipher
- Frequently split the counter into two sections: message number and block number within the message
- Active attacker can make controlled changes to plaintext
- Highly parallelizable; no linkage between stages
- Vital that counter never repeat for any given key
Which Mode for What Task?

- General file or packet encryption: CBC.
- Input must be padded to multiple of cipher block size
- Risk of byte or bit deletion: CFB8 or CFB1
- Bit stream; noisy line and error propagation is undesirable: OFB
- Very high-speed data: CTR
- In most situations, an integrity check is needed

Integrity Checks

- Actually, integrity checks are almost always needed
- Frequently, attacks on integrity can be used to attack confidentiality
- Usual solution: use separate integrity check along with encryption

Combined Modes of Operation

- Integrity checks require a separate pass over the data
- Such checks generally cannot be parallelized
- For high-speed implementations, this is a considerable burden
- Solution: combined modes such as Galois Counter Mode (GCM) or Counter with CBC-MAC (CCM), which provide confidentiality and integrity protection in one pass

Stream Ciphers

- Key stream generator produces a sequence $S$ of pseudo-random bytes; key stream bytes are combined (generally via XOR) with plaintext bytes
- $P_i \oplus S_i \rightarrow C_i$
- Stream ciphers are very good for asynchronous traffic
- Best-known stream cipher is RC4; commonly used with SSL
Stream Ciphers (cont)

- Key stream $S$ must never be reused for different plaintexts:

$$ C = A \oplus K $$

$$ C' = B \oplus K $$

$$ C \oplus C' = A \oplus K \oplus B \oplus K $$

$$ = A \oplus B $$

- Guess at $A$ and see if $B$ makes sense; repeat for subsequent bytes

RC4

- Extremely efficient
- After key setup, it just produces a key stream
- No way to resynchronize except by rekeying and starting over
- Internal state is a 256-byte array plus two integers
- Note: weaknesses if used in ways other than as a stream cipher.

The Guts of RC4

```c
for(counter = 0; counter < buffer_len; counter++) {
    x = (x + 1) % 256;
    y = (state[x] + y) % 256;
    swap_byte(&state[x], &state[y]);
    xorIndex = (state[x] + state[y]) % 256;
    buffer_ptr[counter] ^= state[xorIndex];
}
```

Cipher Strengths

- A cipher is no stronger than its key length: if there are too few keys, an attacker can enumerate all possible keys
- DES has 56 bits — arguably too few in 1976; far too few today.
- Strength of cipher depends on how long it needs to resist attack.
- No good reason to use less than 128 bits
- NSA rates 128-bit AES as good enough for SECRET traffic; 256-bit AES is good enough for TOP-SECRET traffic.
- But a cipher can be considerably weaker! (A monoalphabetic cipher over all bytes has a 1684-bit key, but is trivially solvable.)
### CPU Speed versus Key Size

- Adding one bit to the key doubles the work factor for brute force attacks.
- The effect on encryption time is often negligible or even free.
- It costs *nothing* to use a longer RC4 key.
- Going from 128-bit AES to 256-bit AES takes (at most) 40% longer, but increases the attacker’s effort by a factor of $2^{128}$.
- Using triple DES costs $3 \times$ more to encrypt, but increases the attacker’s effort by a factor of $2^{112}$.
- Moore’s Law favors the defender.

### Pre-Arranged Key Lists?

- What if you run out of keys?
- What if a key is stolen?
  - “Why is it necessary to destroy yesterday’s [key] . . . list if it’s never going to be used again?”
  - “A used key, Your Honor, is the most critical key there is. If anyone can gain access to that, they can read your communications.”
  - (trial of Jerry Whitworth, a convicted spy.)
- What if Alice doesn’t know in advance that she’ll want to talk to Bob?

### Alice and Bob

- Alice wants to communicate security with Bob.
- (Cryptographers frequently speak of Alice and Bob instead of A and B...)
- What key should she use?

### Alice and Bob

- Alice wants to communicate security with Bob.
- (Cryptographers frequently speak of Alice and Bob instead of A and B...)
- What key should she use?

### The Solution: Public Key Cryptography

- Allows parties to communicate without prearrangement.
- Separate keys for encryption and decryption.
- Not possible to derive decryption key from encryption key.
- Permissible to publish encryption key, so that anyone can send you secret messages.
- All known public key systems are very expensive to use, in CPU time and bandwidth.
- Most public systems are based on mathematical problems.
RSA

- The best-known public key system is RSA.
- Generate two large (at least 512 bit) primes $p$ and $q$; let $n = pq$.
- Pick two integers $e$ and $d$ such that $ed \equiv 1 \mod (p-1)(q-1)$. Often, $e = 65537$, since that simplifies encryption calculations.
- The public key is $\langle e, n \rangle$; the private key is $\langle d, n \rangle$.
- To encrypt $m$, calculate $c = m^e \mod n$; to decrypt $c$, calculate $m = c^d \mod n$.
- The security of the system relies on the difficulty of factoring $n$.
- Finding such primes is relatively easy; factoring $n$ is believed to be extremely hard.

Complexities

- RSA calculations are very expensive; neither Bob nor Alice can afford to do many.
- RSA is too amenable to mathematical attacks; encrypting the wrong numbers is a bad idea.
- Example: “yes”³ is only 69 bits, and won’t be reduced by the modulus operation; finding $\sqrt[3]{503565527901556194283}$ is easy.
- We need a better solution

Classical Public Key Usage

- Alice publishes her public key in the phone book.
- Bob prepares a message and encrypts it with that key by doing a large exponentiation.
- Alice uses her private key to do a different large exponentiation.
- It’s not that simple...

A More Realistic Scenario

- Bob generates a random key $k$ for a conventional cipher.
- Bob encrypts the message: $c = \{m\}_k$.
- Bob pads $k$ with a known amount of padding, to make it at least 512 bits long; call this $k'$.
- $k'$ is encrypted with Alice’s public key $\langle e, n \rangle$.
- Bob transmits $\{c, (k')^e \mod n\}$ to Alice.
- Alice uses $\langle d, n \rangle$ to recover $k'$, removes the padding, and uses $k$ to decrypt ciphertext $c$.
- In reality, it’s even more complex than that...
Perfect Forward Secrecy

- If an endpoint is compromised (i.e., captured or hacked), can an enemy read old conversations?
- Example: if an attacker has recorded \{c, (k')^e \mod n\} and then recovers Alice's private key, he can read c.
- Solution: use schemes that provide perfect forward secrecy, such as Diffie-Hellman key exchange.

Diffie-Hellman Key Exchange

- Agree on a large (at least 1024-bit) prime p, usually of the form 2q + 1 where q is also prime.
- Find a generator g of the group "integers modulo p". (This is easy to do if p is prime.)
- Alice picks a large random number x and sends Bob \(g^x \mod p\). Bob picks a large random number y and sends Alice \(g^y \mod p\).
- Alice calculates \(k = (g^y)^x \equiv g^{xy} \mod p\); Bob does a similar calculation.
- If x and y are really random, they can't be recovered if Alice or Bob's machine is hacked.
- Eavesdroppers can't calculate x from \(g^x \mod p\), and hence can't get the shared key. This is called the discrete logarithm problem.

Random Numbers

- Random numbers are very important in cryptography.
- They need to be as random as possible — an attacker who can guess these numbers can break the cryptosystem. (This is a common attack!) To the extent possible, use true-random numbers, not pseudo-random numbers.
- Where do true-random numbers come from?
- Physical processes are best — radioactive decay, thermal noise in amplifiers, oscillator jitter, etc.
- Often, a true-random number is used to seed a cipher — modern cryptographic functions are very good pseudo-random numbers.

Who Sent a Message?

- When Bob receives a message from Alice, how does he know who sent it?
- With traditional, symmetric ciphers, he may know that Alice has the only other copy of the key; with public key, he doesn't even know that.
- Even if he knows, can he prove to a third party — say, a judge — that Alice sent a particular message?
Digital Signatures

- RSA can be used backwards: you can encrypt with the private key, and decrypt with the public key.
- This is a digital signature: only Alice can sign her messages, but anyone can verify that the message came from Alice, by using her public key.
- Again, it’s too expensive to sign the whole message. Instead, Alice calculates a cryptographic hash of the message and signs the hash value.
- If you sign the plaintext and encrypt the signature, the signer’s identity is concealed; if you sign the ciphertext, a gateway can verify the signature without having to decrypt the message.

They’re Not Like Real Signatures

- Real signatures are strongly bound to the person, and weakly bound to the data.
- Digital signatures are strongly bound to the data, and weakly bound to the person — what if the key is stolen (or deliberately leaked)?
- A better term: digital signature algorithms provide non-repudiation.

Cryptographic Hash Functions

- Produce relatively-short, fixed-length output string from arbitrarily long input.
- Computationally infeasible to find two different input strings that hash to the same value.
- Computationally infeasible to find any input string that hashes to a given value.
- Strength roughly equal to half the output length.
- 128 bits and shorter are not very secure for general usage.

Recent Developments

- At CRYPTO ’04, several hash functions were cracked by Wang et al.
- More precisely, collisions were found: $H(M) = H(M')$, $M \neq M'$.
- Cracked functions include MD4, MD5, HAVAL-128, RIPEMD, and SHA-0.
- But SHA-0 was known to be flawed; NSA replaced it with SHA-1 in 1994.
- In 2005, Wang eg al. showed that SHA-1 was considerably weaker than its design strength.
- Are SHA-256/384/512 still secure?
  - MD5 is still commonly used, though weaknesses have long been suspected.
Abusing a Weak Hash Function

- Alice prepares two contracts, \( m \) and \( m' \), such that \( H(m) = H(m') \).
- Contract \( m \) is favorable to Bob; contract \( m' \) is favorable to Alice.
  \( \Rightarrow \) The exact terms aren't important; Alice can prepare many different contracts while searching for two suitable ones.
- Alice sends \( m \) to Bob; he signs it, producing \( \{H(m)\}_{K_B}^{-1} \).
- Alice shows \( m' \) and \( \{H(m)\}_{K_B}^{-1} \) to the judge and asks that \( m' \) be enforced.
- Note that the signature matches.

The Birthday Paradox

- How many people need to be in a room for the probability that two will have the same birthday to be > .5?
- Naive answer: 183
- Correct answer: 23
- The question is not "who has the same birthday as Alice?"; it’s "who has the same birthday as Alice or Bob or Carol or . . . " assuming that none of them have the same birthday as any of the others.

The Birthday Attack

- Alice can prepare lots of variant contracts, looking for any two that have the same hash.
- More precisely, she generates many trivial variants on \( m \) and \( m' \), looking for a match between the two sets.
- This is much easier than finding a contract that has the same hash as a given other contract.
- As a consequence, the strength of a hash function against brute force attacks is approximately half the output block size: 64 bits for MD5, 80 bits for SHA-1, etc.

Birthday Attacks and Block Ciphers

- How many blocks can you encrypt with one key before you start getting collisions?
- The same rule applies: \( 2^{B/2} \) blocks, where \( B \) is the cipher’s block size.
- Thus: \( 2^{32} \) blocks for DES or 3DES; \( 2^{64} \) blocks for AES.
- \( 2^{32} \) 64-bit blocks is \( 2^{35} \) bytes. That’s 34GB — smaller than most modern drives.
- It’s also 275Gb; on a 1Gb/sec network, it’s less than 5 minutes.
- Conclusion: the block size of DES and 3DES is too small for high-speed networks or large disks.
Practical Stunts from MD5 Collisions

- A general style of attack for exploiting hash function collisions has been devised
- Create two prologues to a message file, using a collision assigned to some variable in each version
- In the body of the message, conditionally display one version or the other, depending on that variable
- Get the harmless version signed
- Transmit the harmful one
- If no conditionals in your page description language, put the collision in a font definition file or the like