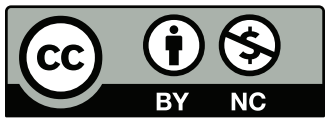
Modes of Operation



Using Cryptography

- As we've already seen, using cryptography properly is not easy
- Many pitfalls!
- Errors in use can lead to very easy attacks
- "You don't go through strong cryptography, you go around it"

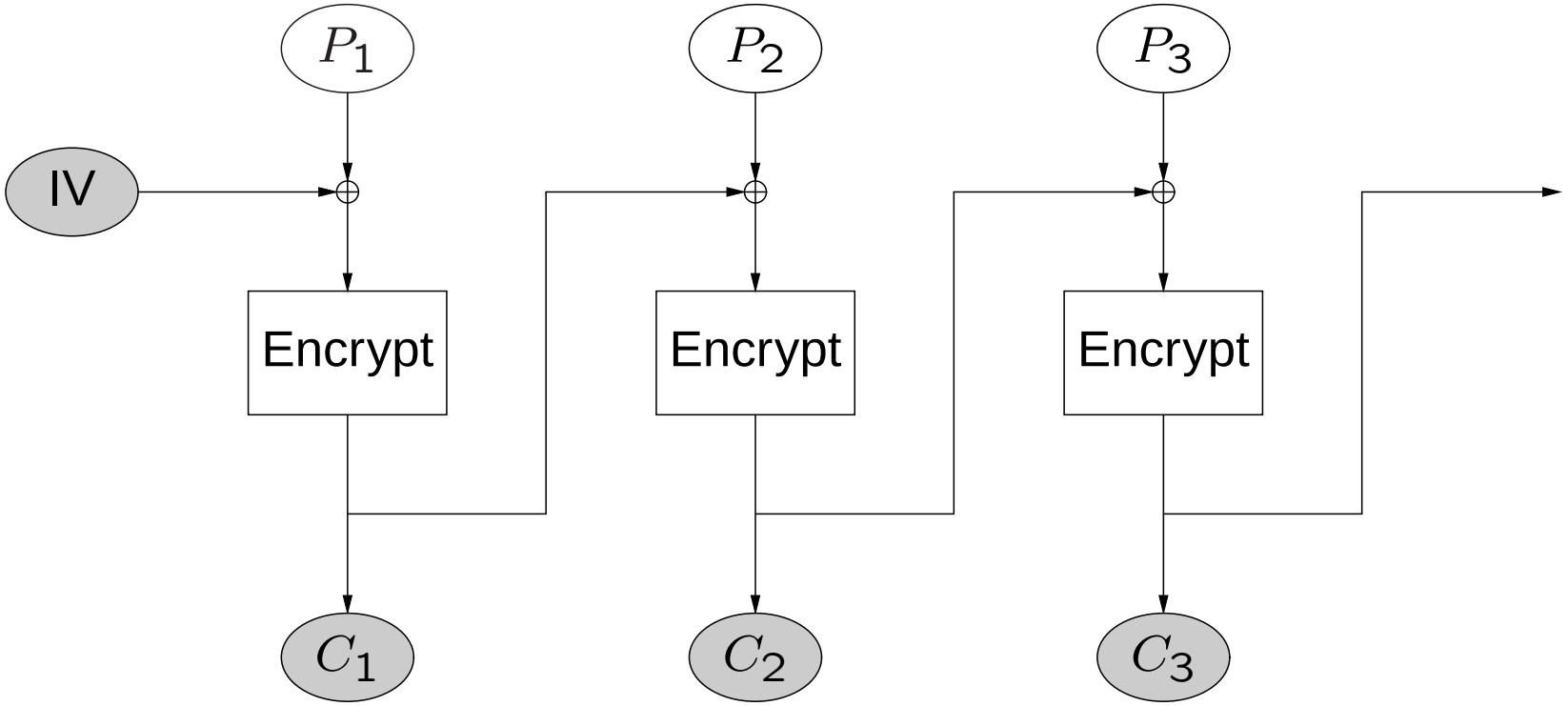
Modes of Operation

- Direct use of a block cipher is inadvisable
- Enemy can build up “code book” of plaintext/ciphertext equivalents
- Beyond that, direct use only works on messages that are a multiple of the cipher block size in length
- Solution: five standard *Modes of Operation*: Electronic Code Book (ECB), Cipher Block Chaining (CBC), Cipher Feedback (CFB), Output Feedback (OFB), and Counter (CTR).

Electronic Code Book

- Direct use of the block cipher
- Used primarily to transmit encrypted keys
- Very weak if used for general-purpose encryption; never use it for a file or a message.
- Attacker can build up codebook; no semantic security
- We write $\{P\}_k \rightarrow C$ to denote "encryption of plaintext P with key k to produce ciphertext C"

Cipher Block Chaining



$$\{P_i \oplus C_{i-1}\}_k \rightarrow C_i$$
$$\{C_i\}_{k^{-1}} \oplus C_{i-1} \rightarrow P_i$$

Properties of CBC

- The ciphertext of each encrypted block depends on the IV and the plaintext of all preceeding blocks.
- There is a dummy initial ciphertext block C_0 known as the *Initialization Vector* (IV); the receiver must know this value.
- Consider a 4-block message:

$$\begin{aligned} C_1 &= \{P_1 \oplus IV\}_k \\ C_2 &= \{P_2 \oplus C_1\}_k \\ C_3 &= \{P_3 \oplus C_2\}_k \\ C_4 &= \{P_4 \oplus C_3\}_k \end{aligned}$$

If C_2 is damaged during transmission, what happens to the plaintext?

Error Propagation in CBC Mode

- Look at the decryption process, where C' is a corrupted version of C:

$$\begin{aligned} P_1 &= \{C_1\}_{k^{-1}} \oplus IV \\ P_2 &= \{C_2'\}_{k^{-1}} \oplus C_1 \\ P_3 &= \{C_3\}_{k^{-1}} \oplus C_2' \\ P_4 &= \{C_4\}_{k^{-1}} \oplus C_3 \end{aligned}$$

- P_1 depends only on C_1 and IV, and is unaffected
- P_2 depends on C_2 and C_1, and hence is corrupted
- P_3 depends on C_3 and C_2, and is also corrupted. The enemy can control the change to P_3.
- P_4 depends on C_4 and C_3, and not C_2; it thus isn't affected.
- Conclusion: Two blocks change, one of them predicatably

Cutting and Pasting CBC Messages

- Consider the encrypted message

$$IV, C_1, C_2, C_3, C_4, C_5$$

- The shortened message IV, C_1, C_2, C_3, C_4 appears valid
- The truncated message C_2, C_3, C_4, C_5 is valid: C_2 acts as the IV.
- Even C_2, C_3, C_4 is valid, and will decrypt properly.
- Any subset of a CBC message will decrypt cleanly.
- If we snip out blocks, leaving IV, C_1, C_4, C_5, we only corrupte one block of plaintext.
- Conclusion: if you want message integrity, you have to do it yourself.

The IV

- The IV provides semantic security: identical messages have different ciphertexts
- Common message start is also hidden
- The IV *must* be unpredictable by the enemy
- Good strategy: have a random key lying around; encrypt a counter to produce the IV
- Note: since the IV is transmitted unencrypted, the other side need not know this key

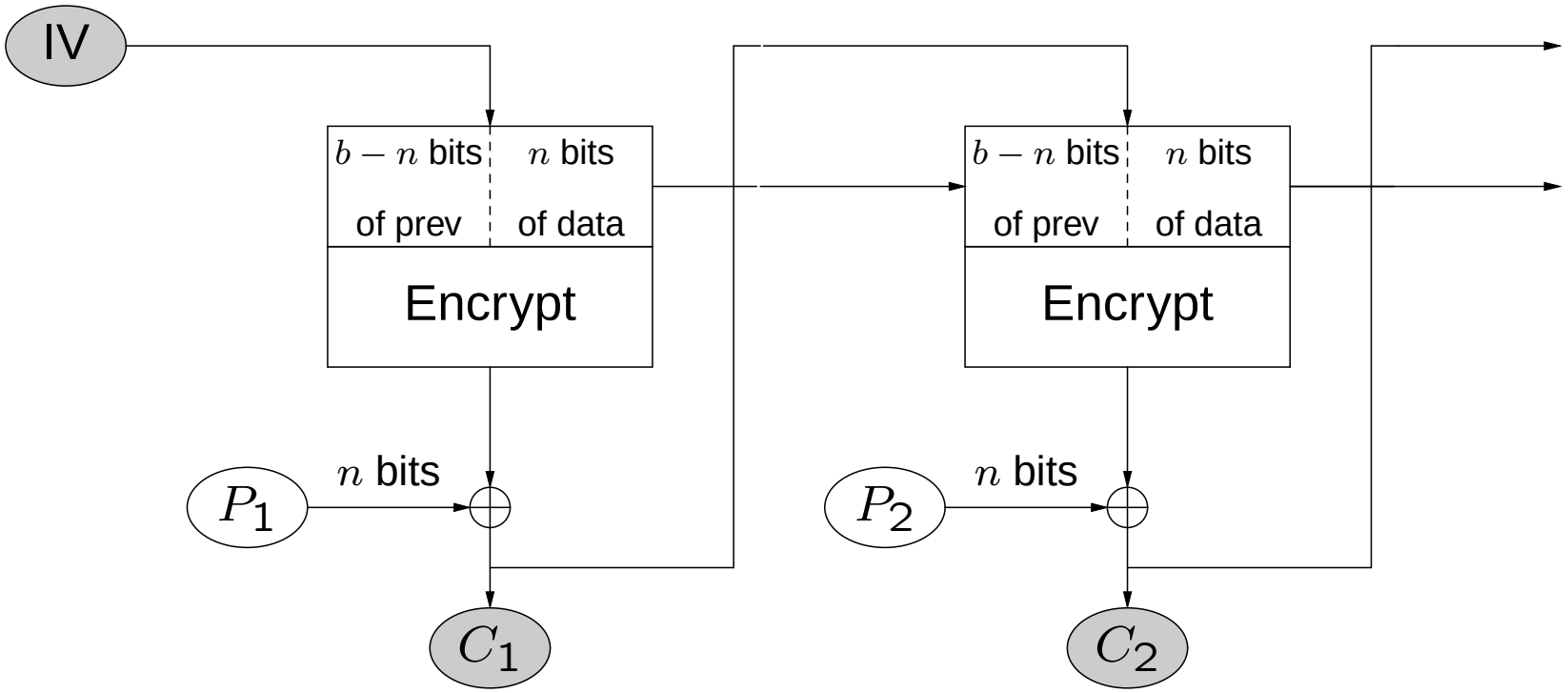
CBC Padding

- CBC mode requires the input to be a multiple of the cipher's block size.
- Must (somehow) handle other lengths
- Usual strategy: (securely) transmit explicit input length
- Option: *Cipher-Text Steaing* (see RFC 2040). Does not increase message size

CBC MAC

- Recall that for CBC encryption, the last block of ciphertext depends on all of the plaintext
- Do a second encryption (using a different key), but only send the last block
- If you use the same key, a CBC cut-and-paste attack will work
- Query: what IV should you use? (It doesn't matter; a constant IV will suffice.)
- Warning: if message sizes can vary, *prepend* the length
- NIST standard CMAC uses slightly more complex scheme that avoids needing to know the length in advance

n-bit Cipher Feedback

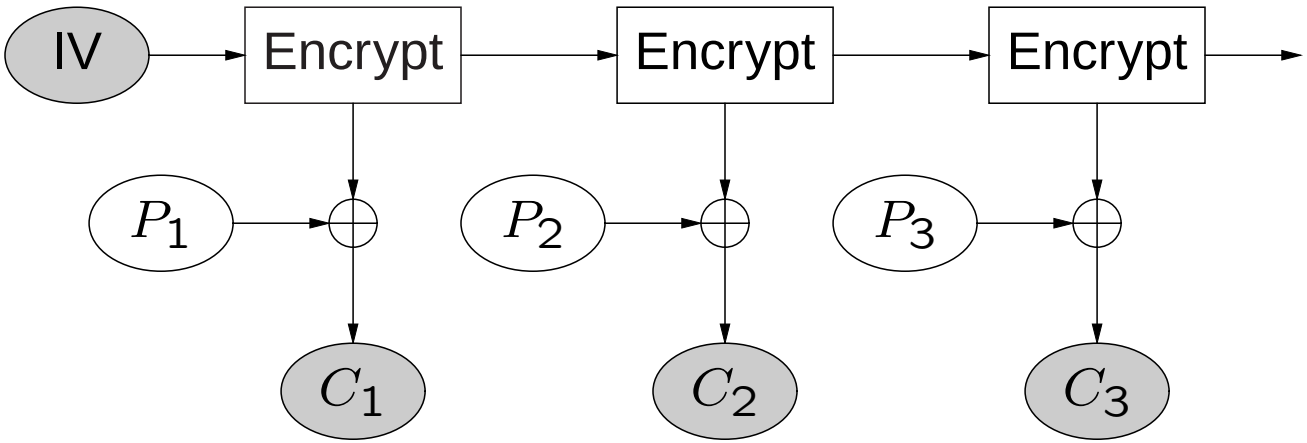


$$P_i \oplus \{C_{i-1}\}_k \rightarrow C_i$$
$$\{C_{i-1}\}_k \oplus C_i \rightarrow P_i$$

Properties of Cipher Feedback Mode

- Underlying block cipher used only in encryption mode
- Feedback path actually incorporates a shift register: shift old cipher input left by n bits; insert first n bits of previous ciphertext output
- 8-bit CFB is good for asynchronous terminal traffic — but requires one encryption for each *byte* of plaintext
- Errors propagate while bad data is in the shift register — 17 bytes for CFB_8 when using AES.
- Copes gracefully with deletion of n-bit unit
- Interesting uses: CFB_1, CFB_8, CFB_128
- IV selected the same way as in CBC mode

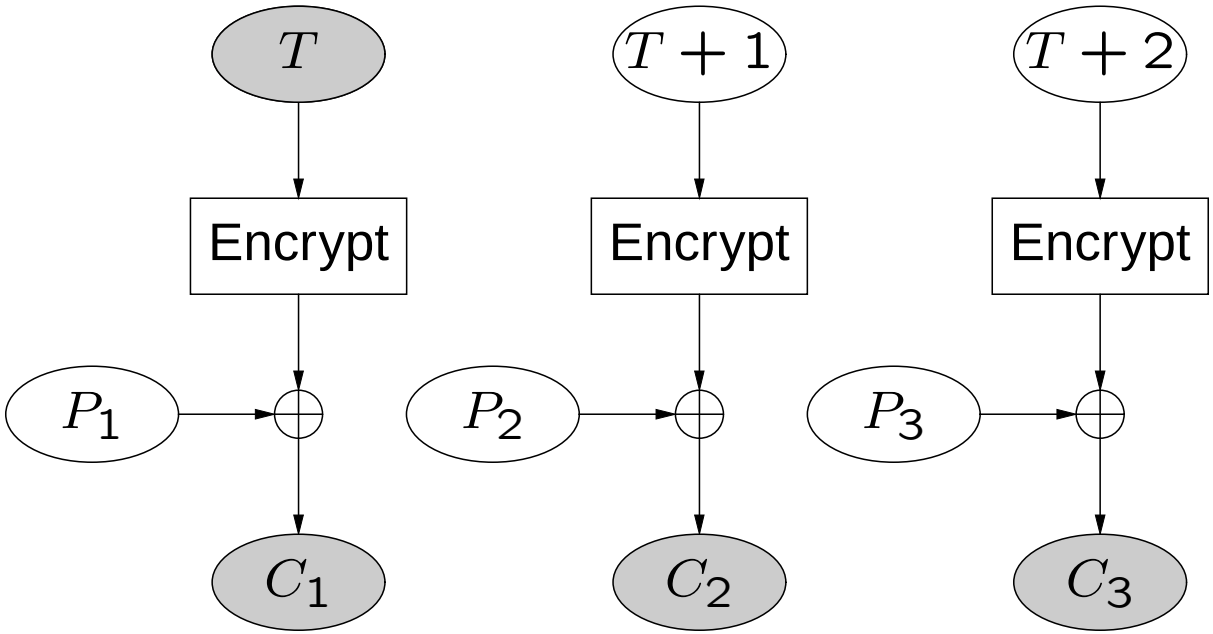
n-bit Output Feedback



Properties of Output Feedback Mode

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

Counter Mode



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: CFB_8 or CFB_1
- 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
- Simple solutions don't work
- Choices: HMAC, CBC MAC, CMAC, combined modes of operation, lesser-known schemes
- One bad idea: append a cryptographic hash to some plaintext, and encrypt the whole thing with, say, CBC mode

$$\{P \parallel H(P)\}_K$$

- This can fall victim to a *chosen plaintext attack*

Chosen Plaintext Attack

- The enemy picks some plaintext P and tricks you into encrypting it
- This has happened in the real world!
- You transmit

$$\{\text{prefix} \parallel P \parallel \text{suffix} \parallel H(\text{prefix} \parallel P \parallel \text{suffix})\}_K$$

- But P is of the form

$$\text{prefix} \parallel P' \parallel \text{suffix} \parallel H(\text{prefix} \parallel P' \parallel \text{suffix})$$

- An ordinary CBC subset will have the checksum!

CCM Mode

- Inputs: nonce N, additional data A (e.g., network packet headers), plaintext P
- Authenticates A and P; encrypts P
- Calculate $T \leftarrow$ CBC-MAC$_K(N \parallel A \parallel P)$
- Ciphertext is CTR$_K(P)$; MAC is CTR$_K(T)$
- Note: the first counter value is used to encrypt T
- Note: N acts as an IV; it must be non-repeating but need not be secret or unpredictable

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

$$\begin{aligned} C &= A \oplus K \\ C' &= B \oplus K \\ C \oplus C' &= A \oplus K \oplus B \oplus K \\ &= A \oplus B \end{aligned}$$

- 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

```
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];
 }
```

Order of Encryption and MACing

- If we want to encrypt and MAC a message, what order do we do it in?
- Three choices:

$$\{P\}_K \parallel M_{K'}(P)$$

$$\{P \parallel M_{K'}(P)\}_K$$

$$\{P\}_K \parallel M_{K'}(\{P\}_K)$$

- The last is the most secure (provably so) — always calculate a MAC on the ciphertext
- Besides, since MACs are often cheaper than decryption, we can verify the integrity of ciphertext first, and discard the message if bogus

What to MAC?

- Obviously, the MAC includes all of the ciphertext
- Frequently, the MAC should include plaintext metadata
- Example: suppose you supply a plaintext message length, in plaintext, to go along with CBC encryption of a padded message
- Example: for network data, may wish to MAC the packet headers

Key Lifetimes

- A confidentiality key is useful as long as the data is sensitive; that may be many years
- A digital signature private key is useful as long as you need to prove authorship — think of a digitally-signed, 30-year mortgage
- A MAC key is useful only while the session is alive; once the session is over, the key is useless

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

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 256 byte values has a 1684-bit key ($\log_2 256! \approx 1684$), 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

Moore's Law and Public Key Cryptography

- For RSA, doubling the modulus length increases encryption time by $4\times$ and decryption time by $8\times$
- Attack time against RSA is based on factoring algorithms, not brute force: there are far too many possible primes for brute force to be ever be possible
- For number field sieve, complexity is approximately proportional to

$$.02e^{1.92\sqrt[3]{\ln n \cdot (\ln \ln n)^2}}$$

- Sub-linear, but space complexity goes up much faster
- There is a paper design for a $10M machine (TWIRL) to factor a single 1024-bit number in one year

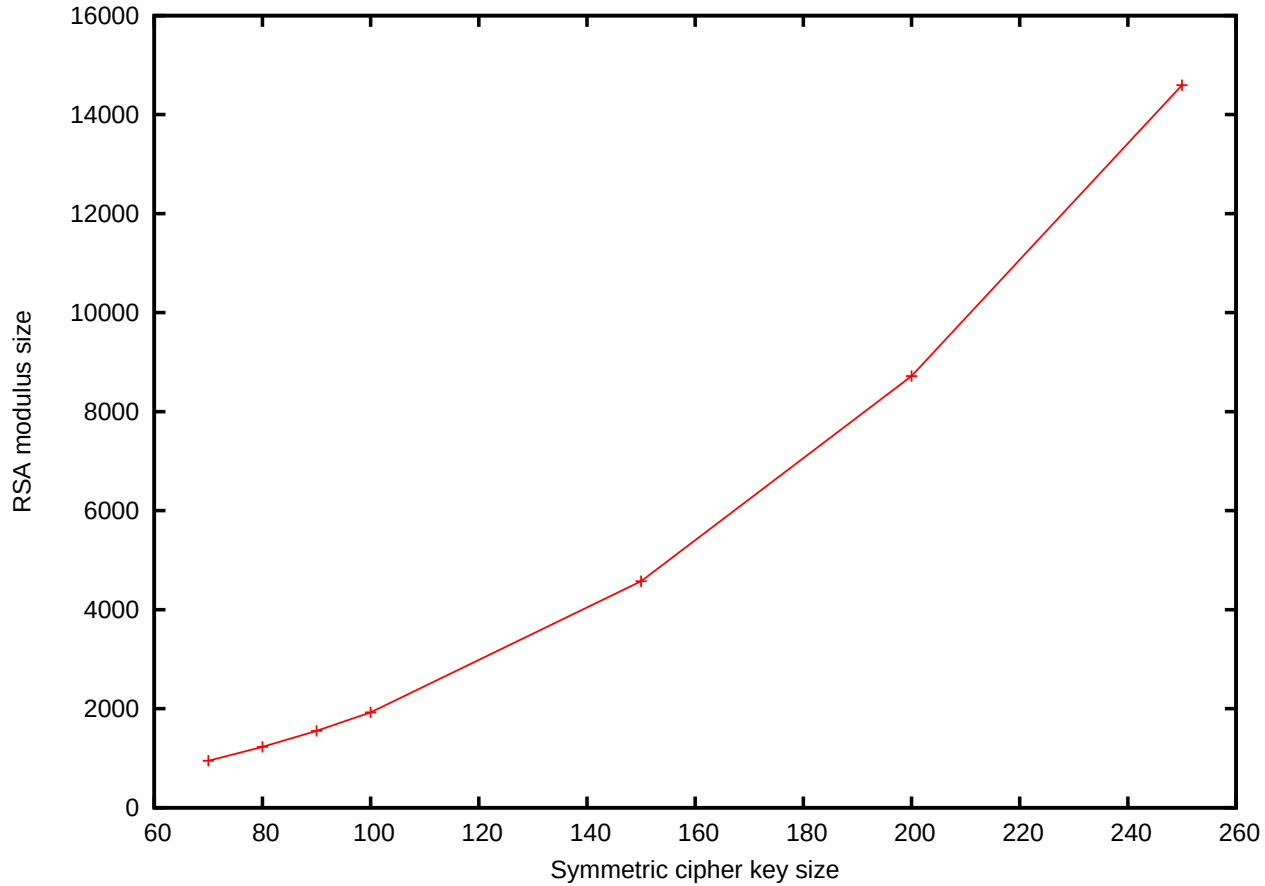
Rough Table of Key Length Equivalences

Symmetric Key Size (bits)	*RSA or DH Modulus Size (bits)*
70	947
80	1228
90	1553
100	1926
150	4575
200	8719
250	14596

Add 11 bits to the public key size if TWIRL can be built

(Numbers by Orman and Hoffman, RFC 3766)

Public versus Symmetric Key Sizes



Current Key Length Recommendations

- For most purposes, 128-bit symmetric keys are sufficient
- For data that must be protected for many years, use 192 bits or 256 bits
- 1024-bit public keys are marginal; new keys should probably be 1536 bits or longer
- Keys for certificate authorities should be at leat 2048 bits and perhaps 3072 bits
- As always, consider your adversary's powers
- "Attacks always get better; they never get worse"

Cipher Design is Hard

- *Never* design your own cipher
- *Never* trust a product that uses a custom cipher design
- Design of ciphers is a difficult, arcane speciality. It takes years of scrutiny by experts to validate a design
- Most of the submissions to the AES competition were cryptanalyzed within a year or two

Hash Function Design is Hard

- Hash functions are *harder* to design
- Even NSA can get it wrong: SHA-0 was replaced after two years by SHA-1, but they differ only in a single circular shift
- SHA-0 has been cracked; SHA-1 is (as stated) weak, but no collisions have been found yet

Modes of Operation are Hard

- There have been many published — and implemented — modes that have been broken
- “Obviously” stronger modes may not be (a variant on DES CBC MAC that should have had 112 bit strength had 57-bit strength)
- Some variations are better, but most are not

A Note on 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 a common attack!) To the extent possible, use true-random numbers, not pseudo-random numbers — but watch out for low-assurance generators
- 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.