CPSC/PMAT 669

Product Ciphers, Block Ciphers, AES

Mike Jacobson

Department of Computer Science
University of Calgary

Topic 3

Outline

Product Ciphers

Shannon also introduced the idea of product ciphers (multiple encryption):

Definition 1 (Product cipher)
The *product* of two ciphers is the result of applying one cipher followed by the other.

AKA *superencipherment* and various other names.

Properties of Product Ciphers

If different ciphers are used in a product cipher, ciphertexts of one cipher need to have the correct format to be plaintexts for the next cipher to be applied.

- This is composition of encryption maps.

Applying a product cipher potentially increases security. E.g. n-fold encryption with one cipher and n keys potentially corresponds to a cipher that has n times longer keys.

Of course it also results in a loss of speed by a factor of n, but this might be worth it for added security.

Caveat

Be careful with this reasoning!

Note 1

The product of two substitution ciphers is a substitution cipher. The product of two transposition ciphers is a transposition cipher.

Such ciphers are closed under encryption, so multiple encryption under different keys provides no extra security:

E.g. double encryption $C = E_{K_1}(E_{K_2}(M)) = E_{K_3}(M)$ for a third key K_3

Confusion and Diffusion

Shannon suggested applying two simple ciphers with a fixed mixing transformation (transposition) in between to:

- *diffuse* language redundancy into long term statistics, and to
- *confuse* the cryptanalyst by making relation between redundancy of the ciphertext C and the description of the key K very complex.

Definition 2 (Confusion)

Make the relationship between the key and ciphertext as complex as possible (accomplished by applying substitutions or *S-boxes*).

Definition 3 (Diffusion)

Dissipate the statistical properties of the plaintext across the ciphertext (accomplished by applying applying transpositions or *P-boxes*).

Examples of Product Ciphers

- ADFGX/ADFGVX Ciphers (employed by the Germans in WW I).
- Hayhanen Cipher — Hayanan was a Russian spy caught in New York in the 1950's. The FBI couldn't break it!

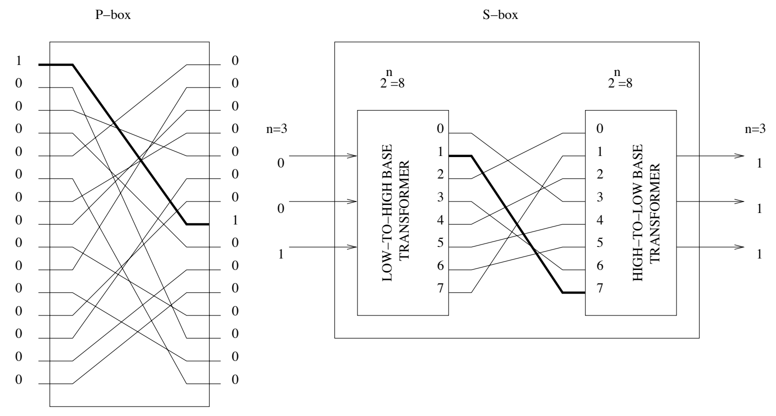
Example 4

IBM's Lucifer system, uses permutations (transpositions) on large blocks for the mixing transformation, and substitution on small blocks for confusion.

Feistel originally wanted to call the product cipher "Dataseal". IBM instead shortened the term *demonstration cipher* to "Demon." Later, it was changed to *Lucifer*, because it retained the "evil atmosphere" of Demon, and contained the word *cipher*.

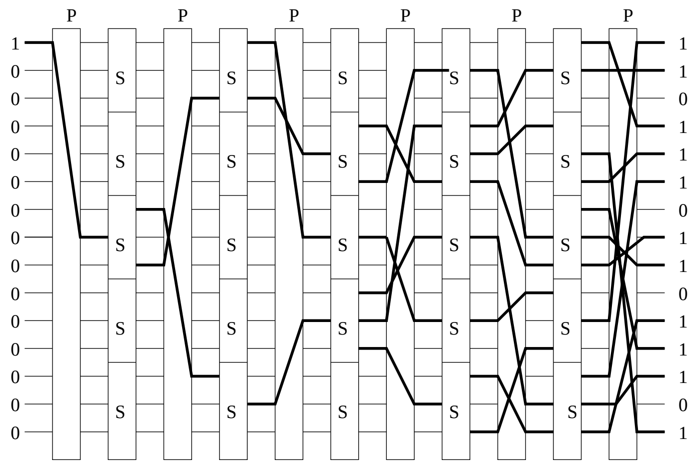
Lucifer: P-boxes and S-boxes

Since Lucifer was set up in hardware, they called the chips which did the permutation "P-boxes" and those that did the substitution "S-boxes."



Lucifer: Complete Cipher

The Lucifer system simply consisted of a number of P and S boxes in alternation.



Error Propagation

Lucifer demonstrates graphically not only the difffusion properties of Lucifer, but also the idea of *error propagation*.

Definition 5 (Error Propagation)

The degree to which a change in the input leads to changes in the output.

Good error propagation is a desirable property of a cryptosystem (a user can easily tell if a message has been modified).

Not good for decryption though (one error in the process should still mostly decrypt correctly).

Error Propagation in Lucifer

Example 6

Lucifer has good diffusion and error propagation. The thicker lines in the graphic indicate ‘1’ bits. The ‘1’ input bit dissipates over the entire ciphertext.

Aso, if the first ‘1’ bit is changed to a ‘0’, then half the bits in the output are affected.

Note: All modern symmetric key ciphers in use are product ciphers.

Block Ciphers

All modern ciphers in use are block ciphers (although not necessarily used as such — we’ll talk about modes of operation of block ciphers later).

Definition 7 (Block cipher)

Encrypts plaintext blocks of some fixed length to ciphertext blocks of some fixed (possibly different) length.

Usually, a message M will be larger than the plaintext block length, and must hence be divided into a series of sequential message blocks $M_1, M_2, \dots, M_n$ of the desired length.

- A block cipher operates on these blocks one at a time.

Examples of Block Ciphers

Example 8

The shift cipher is a block cipher where the blocks consists of one character (*i.e.* 8 bits on 32-bit architecture, 16 bits on 64-bit architecture).

Two main block ciphers in use today:

1. Data Encryption Standard (DES)
   - 64-bit plaintext blocks, 64-bit ciphertext blocks, 64-bit keys (56 actual key bits, 8 parity)
   - obsolete, so now used in triple encipherment as 3DES
2. Advanced Encryption Standard (AES)
   - 128-bit plaintext and ciphertext blocks, 128, 192, or 256-bit keys
   - Current NIST-endorsed standard, widely used

NIST Publications

- NIST: National Institute of Standards and Technology (formerly National Bureau of Standards (NBS))
- FIPS: Federal Information Processing Register

Everything about NIST's cryptographic standards, recommendations, and guidance can be found at the NIST Cryptographic Toolkit website `http://csrc.nist.gov/CryptoToolkit`.

- Extremely useful website for the practicing cryptographer.
- There is a link on the "external links" page on the course website.

History of DES

Data encryption standard:

- DES was developed by IBM around 1972.
- NBS made solicitations to IBM in 1973/1974 concerning the use of this as a public standard, in response to corporate needs for securing information.
- IBM and the National Security Agency (NSA) secretly evaluated DES for security.
- DES was approved in 1978, and it is still in use, although no longer endorsed.

Current DES publication: FIPS 46-3 (October 25, 1999)

Assessment of DES

Great cipher at the time (efficient, good for hardware, same encryption/decryption algorithm, withstood all known attacks)

The problem is that for today's computers, the key space of DES is simply too small to foil exhaustive search ($2^{56} \approx 10^{17}$ keys).

- In 1998, the *Electronic Frontier Foundation* built a *DES Cracker* for \$250,000 which finds a single DES key in 56 hours (tests 8800 keys/μ-sec).
- A combination of the DES cracker and 100,000 PCs on the internet has found a DES key in 22.25 hours (tested 245,000 keys/μ-sec).

Multiple DES Encryption

What about multiple DES encryptions? Does this foil exhaustive attacks due to longer key sizes?

Campbell and Wiener (1992) proved that DES is *not* closed, so multiple DES encryptions/decryptions could potentially provide additional security.

- size of the group generated by all the keys (*i.e.* the number of distinct encryptions obtained by applying repeated DES encryptions) has been shown to have size at least $10^{2499} \approx 2^{8302}$. (Estimated number of atoms in the universe: 2^{240}.)

Later, we will show that on double encryption is essentially no more secure than single encryption (but twice as slow).

What about three DES encryption? 3DES (triple DES) is still used.

Triple DES

Use three successive DES operations: $C = DES_{K_1}(DES^{-1}_{K_2}(DES_{K_3}(M)))$

- See NIST Special Publication SP 800-67.

Advantages:

- Same as single key if $K_2 = K_1$ or $K_2 = K_3$.
- Exhaustive search has complexity 2^{112} via the meet-in-the-middle attack (see next week), but with a 168-bit key and a factor of 3 in speed.
- Can use $K_1 = K_3$ with no loss of security.
- No other known practical attacks.

The main disadvantage is that 3-DES is three times slower than single key DES while only doubling the key size.

Skipjack and the Clipper Chip

After DES became obsolete, the United States *National Security Agency* (NSA) wanted to take control of the cipher standard selection process

- Proposed the *Skipjack Algorithm* implemented on the *Clipper Chip*
- Standardized by NIST as Escrowed Encryption Standard (EES) in Feb. 1994 (see FIPS 185).

In blunt violation of Kerckhoff's principle, the details of Clipper and Skipjack were initially classified and kept secret.

Due to wide distrust of the NSA in the US and abroad, this never really caught on in the public sector.

AES Competition

In 1997, NIST put out a call soliciting candidates to replace DES using a process that was completely transparent and public. Requirements:

- possible key sizes of 128, 192, and 256 bits
- plaintexts and ciphertexts of 128 bits
- should work on a wide variety of hardware (from Smart Cards to PCs)
- fast
- secure
- world-wide royalty-free availability (!)

Selection Criteria

Candidates were selected according to:

- security – resistance against all known attacks
- cost — speed and code compactness on a wide variety of platforms
- simplicity of design

Most important: *public* evaluation process

- series of three conferences: algorithms, attacks, evaluations presented and discussed
- final selection done by NIST

Finalists

NIST initiated a public (world-wide) process of candidate submission and evaluation for the *Advanced Encryption Standard*.

21 algorithms were submitted on June 15, 1998, of which 15 were announced as candidates on August 20, 1998.

Five finalists were selected in August 1999:

- MARS (from IBM)
- RC6 (from RSA Labs)
- Rijndael (by two Belgians: Daemen and Rijmen)
- Serpent (Anderson, Biham, Knudson, a multi-national team)
- Twofish (Schneier et al)

The Winner: Rijndael

All five were very good algorithms. Rijndael (pronounced “Reign Dahl” or “Rhine Dahl”, but NOT “Region Deal”) was chosen as the AES.

- Inventors: Vincent Rijmen and Joan Daemen.

The Rijndael algorithm uses two different types of arithmetic:

- Arithmetic on bytes (8 bit vectors—actually, elements of the finite field $GF(2^8)$ of 256 elements)
- 4-byte vectors (actually polynomial operations over $GF(2^8)$).

Arithmetic on Bytes

Consider a byte $b = (b_7, b_6, \dots, b_1, b_0)$ (an 8-bit vector) as a polynomial with coefficients in $\{0, 1\}$:

$$b \mapsto b(x) = b_7x^7 + b_6x^6 + \cdots + b_1x + b_0 \ .$$

Byte operations (interpretted as polynomials):

1. Addition (component-wise addition, i.e., XOR)
2. Multiplication modulo $m(x) = x^8 + x^4 + x^3 + x + 1$
3. Inversion modulo $m(x)$

Under these operations, polynomials of degree ≤ 7 with coefficients in $\{0, 1\}$ form the *field* $GF(2^8)$.

By associating bytes with these polynomials, we obtain these operations on bytes (for ByteSub).

Arithmetic on 4-byte Vectors

In Rijndael's MixColumn operation, 4-byte vectors are considered as degree 3 polynomials with coefficients in $GF(2^8)$. That is, the 4-byte vector (a_3, a_2, a_1, a_0) is associated with the polynomial

$$a_3x^3 + a_2x^2 + a_1x + a_0,$$

where each coefficient is a byte viewed as an element of $GF(2^8)$

Operations:

- addition: component-wise "addition" of coefficients (addition as described above)
- multiplication: polynomial multiplication (addition and multiplication of coefficients as described above) modulo $M(x) = x^4 + 1$. Result is a degree 3 polynomial with coefficients in $GF(2^8)$.

Literature

Website: `http://csrc.nist.gov/CryptoToolkit/aes/rijndael/`

AES Description:
`http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf`

See also `http://www.iaik.tu-graz.ac.at/research/krypto/AES/` for other information (ECRYPT network).

Rijndael Properties

Designed for block sizes and key lengths to be any multiple of 32, including those specified in the AES.

Iterated cipher: number of rounds N_r depends on the key length. 10 rounds for 128 bit keys, 12 rounds for 192 bit keys, and 14 rounds for 256 bit keys.

Algorithm operates on a 4×4 array of bytes (8 bit vectors) called the *state*:

$s_{0,0}$	$s_{0,1}$	$s_{0,2}$	$s_{0,3}$
$s_{1,0}$	$s_{1,1}$	$s_{1,2}$	$s_{1,3}$
$s_{2,0}$	$s_{2,1}$	$s_{2,2}$	$s_{2,3}$
$s_{3,0}$	$s_{3,1}$	$s_{3,2}$	$s_{3,3}$

Properties, cont.

The algorithm uses addition, multiplication, and inversion on bytes as well as addition and multiplication of 4 byte vectors.

Rijndael is a product cipher, but NOT a Feistel cipher like DES. Instead, it has three *layers* per round:

- a linear mixing layer (ShiftRows, transposition, and MixColumns, a linear transformation; for diffusion over multiple rounds)
- a non-linear layer (SubBytes, substitution, done with an S-box)
- a key addition layer (AddRoundKey, X-OR with key)

Overview

The Rijndael algorithm (given plaintext M) proceeds as follows:

1. Initialize `State` with M :

$$\begin{array}{|c|c|c|c|}\hline s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ \hline s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\ \hline s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\ \hline s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} \\ \hline\end{array} \leftarrow \begin{array}{|c|c|c|c|}\hline m_0 & m_4 & m_8 & m_{12} \\ \hline m_1 & m_5 & m_9 & m_{13} \\ \hline m_2 & m_6 & m_{10} & m_{14} \\ \hline m_3 & m_7 & m_{11} & m_{15} \\ \hline\end{array}$$

   where M consists of the 16 bytes $m_0, m_1, \ldots, m_{15}$.

Overview, cont.

2. Perform ADDROUNDKEY, which X-OR's the first `RoundKey` with `State`.
3. For each of the first $N_r - 1$ rounds:
   - Perform SUBBYTES on `State` (using an S-box on each byte of `State`),
   - Perform SHIFTROWS (a permutation) on `State`,
   - Perform MIXCOLUMNS (a linear transformation) on `State`,
   - Perform ADDROUNDKEY.
4. For the last round:
   - Perform SUBBYTES,
   - Perform SHIFTROWS,
   - Perform ADDROUNDKEY.
5. Define the ciphertext C to be `State` (using the same byte ordering).

The SUBBYTES Operation

Each byte of `State` is substituted independently, using an invertible S-box (see p. 16 of FIPS 197 for the exact S-Box).

Algebraically, SUBBYTES performs on each byte:

- an inversion as described above (the inverse of the zero byte is defined to be zero here), followed by
- an affine transformation, *i.e.* a linear transformation (like in linear algebra), and the addition of a fixed vector. More exactly, the i=th bit of the output byte is

$$b_i' = b_i \oplus b_{i+4 \bmod 8} \oplus b_{i+5 \bmod 8} \oplus b_{i+6 \bmod 8} \oplus b_{i+7 \bmod 8} \oplus c_i$$

  where b_i is the i-th input bit and c_i is the i-th bit of $c = (11000110)$.

Inverse of SUBBYTES

The inverse of SUBBYTES (called INVSUBBYTES) applies the inverse S-box to each byte `State` (see p. 22 of FIPS 197 for the inverse of the S-Box).

Algebraically, you first apply the inverse affine transformation to each bit and then byte inversion.

The ShiftRows Operation

Shifts the first, second, third, and last rows of `State` by 0, 1, 2, or 3 cells to the left, respectively:

$$\begin{array}{|c|c|c|c|}\hline s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ \hline s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\ \hline s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\ \hline s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} \\ \hline \end{array} \leftarrow \begin{array}{|c|c|c|c|}\hline s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ \hline s_{1,1} & s_{1,2} & s_{1,3} & s_{1,0} \\ \hline s_{2,2} & s_{2,3} & s_{2,0} & s_{2,1} \\ \hline s_{3,3} & s_{3,0} & s_{3,1} & s_{3,2} \\ \hline \end{array}$$

The inverse operation InvShiftRows applies right shifts instead of left shifts.

The MixColumns Operation

Each column of `State` is a 4-byte vector which can be interpreted as a four-term polynomial with coefficients in $GF(2^8)$ as described above. For example:

$$(s_{0,0}, s_{1,0}, s_{2,0}, s_{3,0}) \mapsto s_{3,0}x^3 + s_{2,0}x^2 + s_{1,0}x + s_{0,0} = col_0(x) \ .$$

Let $a(x) = 3x^3 + x^2 + x + 2$ be fixed.

Then MixColumns multiplies $col_i(x)$ by $a(x)$ as described above (multiplication of two 4-byte vectors), resulting in a new 4-byte column.

MixColumns: Algebraic Description

MixColumns can also be described as a linear transformation applied to each column of `State`, *i.e.* multiplying each 4-element column vector by the 4×4 matrix.

$$\begin{pmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 1 \end{pmatrix}$$

Note that rows 0, 1, 2, 3 of this matrix are circular shifts of row 0 by 0, 1, 2, 3 cells to the right.

InvMixColumns: Algebraic Description

The inverse (called InvMixColumns) multiplies each column of `State` by the inverse of $a(x) \pmod{x^4+1}$ which is

$$a^{-1}(x) = Bx^3 + Dx^2 + 9x + E$$

in hex notation.

It can also be described as multiplication by the following matrix (in hex):

$$\begin{pmatrix} E & B & D & 9 \\ 9 & E & B & D \\ D & 9 & E & B \\ B & D & 9 & E \end{pmatrix}$$

The AddRoundKey Operation

In AddRoundKey, each column of `State` is X-ORed with one word of the round key:

$$\begin{array}{|c|c|c|c|}\hline s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ \hline s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\ \hline s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\ \hline s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} \\ \hline \end{array} \leftarrow \begin{array}{|c|c|c|c|}\hline s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ \hline s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\ \hline s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\ \hline s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} \\ \hline \end{array} \oplus \begin{array}{|c|c|c|c|}\hline w_{0,i+0} & w_{0,i+1} & w_{0,i+2} & w_{0,i+3} \\ w_{1,i+0} & w_{1,i+1} & w_{1,i+2} & w_{1,i+3} \\ w_{2,i+0} & w_{2,i+1} & w_{2,i+2} & w_{2,i+3} \\ w_{3,i+0} & w_{3,i+1} & w_{3,i+2} & w_{3,i+3} \\ \hline \end{array}$$

Here $w_{i+0} = (w_{0,i+0}, w_{1,i+0}, w_{2,i+0}, w_{3,i+0})$ is the first round key for round i, made up of four bytes.

AddRoundKey is clearly its own inverse.

Key Schedule

The key schedule uses:

- the S-box from `SubBytes`
- cyclic left shifts by one byte on 4-byte vectors
- multiplication by powers of x (each such power is interpreted as a 4-byte vector)

Consider 128-bit Rijndael. There are 10 rounds plus one preliminary application of AddRoundKey, so the key schedule must produce 11 round keys, each consisting of four 4-byte words, from the 128-bit key (16 bytes).

KeyExpansion

Produces an expanded key consisting of the required 44 words (assuming 128-bit key).

In the following, the key $K = (k_0, k_1, k_2, k_3)$, where the k_i are 4-byte words, and the expanded key is denoted by the word-vector $(w_0, w_1, w_2, \ldots, w_{44})$.

1. for $i \in \{0, 1, 2, 3\}$, $w_i = k_i$
2. for $i \in \{4, \ldots, 44\}$:

$$w_i = w_{i-4} \oplus \begin{cases} \textsc{SubWord}(\textsc{RotWord}(w_{i-1})) \oplus \textsc{Rcon}_{i/4} & \text{if } 4 \mid i \\ w_{i-1} & \text{otherwise} \end{cases}$$

KeyExpansion, cont.

The components of KeyExpansion are:

- RotWord is a one-byte circular left shift on a word.
- SubWord performs a byte substitution (using the S-box SubBytes on each byte of its input word).
- Rcon is a table of round constants ($\textsc{Rcon}_j$ is used in round j). Each is a word with the three rightmost bytes equal to 0 and the leftmost byte a power of x

KeyExpansion is similar for 192 and 256-bit keys.

Decryption

To decrypt, perform cipher in reverse order, using inverses of components and the reverse of the key schedule:

1. ADDROUNDKEY with round key N_r
2. For rounds $N_r - 1$ to 1 :
   - INVSHIFTROWS
   - INVSUBBYTES
   - ADDROUNDKEY
   - INVMIXCOLUMNS
3. For round 1 :
   - INVSHIFTROWS
   - INVSUBBYTES
   - ADDROUNDKEY using round key 1

Note 2

Straightforward inverse cipher has a different sequence of transformations in the rounds. It is possible to reorgainize this so that the sequence is the same as that of encryption (see A2 of FIPS-197).

Strengths of Rijndael

Secure against all known attacks at the time; some newer attacks seem to pose no real threat

Non-linearity resides in S-boxes (SUBBYTES):

- linear approximation and difference tables are close to uniform (thwarting linear and differential cryptanalysis)
- no fixed points ($S(a) = a$) or opposite fixed points ($S(a) = \bar{a}$)
- not an involution ($S(S(a)) \neq a$, or equivalently, $S(a) = S^{-1}(a)$)

SHIFTROWS and MIXCOLUMNS ensure that after a few rounds, all output bits depend on all input bits (great diffusion).

Strengths, cont.

Secure key schedule (great confusion):

- knowledge of part of the cipher key or round key does not enable calculation of many other round key bits
- each key bit affects many round key bits

Very low memory requirements

Very fast (hardware and software)

Weaknesses of Rijndael

Decryption is slower than encryption.

Decryption algorithm is different from encryption (requires separate circuits and/or tables).

- Depending on the mode of operation, however, this may not be an issue (*i.e.* OFB, CTR, CFB).

Security of AES

There is no mathematical proof that AES is secure

All we know is that in practice, it withstands all modern attacks.

Next: overview of modern attacks on block ciphers

Exhaustive Search

Set $N = |\mathcal{K}|$ (number of keys).

Simple exhaustive search (COA) — requires $|\mathcal{K}|$ encryptions

- feasible for DES — $N = 2^{56} \approx 10^{17}$ possible keys.
- infeasible for 3DES – $N = 2^{112} \approx 10^{34}$ possible key combinations.
- infeasible for AES – $N = 2^{128} \approx 10^{38}$ possible keys

Parallelism can speed up exhaustive search.

Perspective: there are approximately 10^{40} water molecules in Lake Ontario. 10^{38} is significantly bigger than the number of water molecules in Lake Louise or in the stretch of the Bow River through Calgary!

Improvement for DES

Exhaustive search for DES can be cut in half (i.e. to 2^{55} test encryptions) via the property

$$C = E_K(M) \quad \Longrightarrow \quad E_{\overline{K}}(\overline{M}) = \overline{C} \ ,$$

where $\overline{X}$ denotes the *one's complement* of a bit string X (*i.e.* each bit in X is flipped to obtain $\overline{X}$).

Mount a CPA as follows: choose two pairs $(M, C_1 = E_K(M))$ and $(\overline{M}, C_2 = E_K(\overline{M})$. For each test key K', if

- $E_{K'}(M) = C_1$, then $K = K'$,
- $E_{K'}(M) = \overline{C}_2$, then $K = \overline{K'}$, since $E_{K'}(M) = \overline{C}_2 \Rightarrow E_{\overline{K'}}(\overline{M}) = C_2$.

Hellman's Time-memory tradeoff (1980)

KPA that shortens search time by using a lot of memory.

- The attacker knows a plaintext/ciphertext pair (M_0, C_0).
- The goal is to find the (or a) key K such that $C_0 = E_K(M_0)$.

Let $N = |\mathcal{K}|$. Cost (# of encryptions) is

Precomputation time:	N
Expected time:	$N^{2/3}$
Expected memory:	$N^{2/3}$

Large precomputation time, but improvement for individual keys

- For DES, $N^{2/3} \approx 10^{12}$ — can be done in hours or even minutes on a modern computer.

Meet-in-the-Middle Attack

KPA on double encryption.

Setup:

- Adversary has two known plaintext/ciphertexts pairs (m_1, c_1) and (m_2, c_2)
- Double-encryption, so $c_i = E_{k_1}(E_{k_2}(m_i))$ for $i = 1, 2$ and two unknown keys k_1, k_2.

Important observation: $D_{k_1}(c_i) = E_{k_2}(m_i)$ $(i = 1, 2)$.

The Attack

The adversary proceeds as follows:

1. Single-encrypt m_1 under every key K_i to compute $C_i = E_{K_i}(m_1)$ for $1 \le i \le N$.
2. Sort the table (or create a hash table).
3. For $j = 1$ to N do
    1. Single-decrypt c_1 under every key K_j to compute $M_j = D_{K_j}(c_1)$.
    2. Search for M_j in the table of C_i. If $M_j = C_i$ for some i, then check if $E_{K_i}(m_2) = D_{K_j}(c_2)$. If this holds, then guess $k_2 = K_i$ and $k_1 = K_j$ and quit.

Analysis

There are at most N values $E_{K_i}(m_1)$ and at most N values $D_{K_j}(c_1)$ for $1 \le i, j \le N$.

- Assuming random distribution, the chances of a match are $1/N$.
- Thus, $(N \cdot N)/N = N$ key pairs (K_i, K_j) satsify $E_{K_i}(m_1) = D_{K_j}(c_1)$.

The chances that such a key pair also satisfies $E_{K_i}(m_2) = D_{K_j}(c_2)$ are very small (paranoid users could try a third message/ciphertext pair (m_3, c_3)).

Thus, the probability of guessing correctly is very high.

Analysis, cont.

Time required:

- Step 1: N encryptions
- Step 2: sorting/hash table creation is negligible compared to Step 1
- Step 3 (a): at most N decryptions
- Step 3 (b): negligible in light of Step 2

Total: $2N$ encryptions/decryptions.

Memory: N keys and corresponding ciphertexts (the table of (C_i, K_i) pairs)

Conclusion: double encryption offers little extra protection over single encryption (hence 3DES instead of 2DES).

Linear Cryptanalysis

M. Matsui, EUROCRYPT 1993 – CCA

- Matsui actually used this method to become the first person to recover a DES key (50 days using 12 workstations).

Definition 9

A cryptosystem is *affine (linear)* if for all plaintexts M and keys K,

$$C = E_K(M) = AM + BK + H$$

where A and B are matrices and H is a vector of appropriate dimension (A, B and H are public). The system is *linear* if $H = 0$.

Note that B may or may not be square.

Example (DES)

If DES were affine, we would have the following matrix sizes:

$$A : 64 \times 64, \quad B : 64 \times 56,$$
$$K : 56 \times 1, \quad M : 64 \times 1, \quad C : 64 \times 1$$

Examples of affine linear cryptosystems are:

- the shift cipher
- the Vigenére cipher
- any transposition cipher
- the one-time pad.

Attacking Linear Cryptosystems

A cryptanalyst knowing a plaintext/ciphertext pair (M, C) can easily mount a KPA on an affine or linear system as follows:

$$\begin{aligned} BK &= C - AM - H \\ B^T BK &= B^T(C - AM - H) \\ K &= (B^T B)^{-1} B^T (C - AM - H) \end{aligned}$$

Idea of Linear Cryptanalysis

If a cryptosystem is “close to” being affine then the modified system can be broken and original system compromised after some searching.

- “close to affine” if modifying a few entries in the system (eg. in the S-boxes) makes it affine on certain plaintext/ciphertext pairs

Linear cryptanalysis attempts to “linearly approximate” non-linear cryptosystems in this way.

Every building block in DES and AES *except the S-boxes* is affine.

- S-boxes *must not* be “close” to linear (*i.e.* closely approximated by a linear function).

Differential cryptanalysis

Biham and Shamir, Journal of Cryptology, 1991 — KPA

Compares input XORs to output XORs, and traces these differences through the cipher.

Both linear and differential cryptanalysis work quite well on DES with fewer than 16 rounds.

- The first edition of Stinson's book (1995) discusses successful differential cryptanalysis attacks on 3-round and 6-round DES.
- Large-scale, parallel, brute-force attack is still the most practical attack on 16 round DES.

Requirements for full DES

Type of attack	Expected time	# of (M, C) pairs
Exhaustive search	2^{55}	none
Linear Cryptanalysis	2^{43}	2^{43} (chosen)
Differential Cryptanalysis	2^{47}	2^{47} (known)

Note: AES not affected by these attacks (by design)

Algebraic Attacks

Courtois 2001 — KPA, generates multivariate equations from S-boxes, where the unknowns are the key bits.

- So far no threat to any modern block cipher.

Obstacle: solving multivariate equations seems to be hard in practice

Biclique Attack

Enhanced meet-in-the-middle attack using *bicliques* that map internal states to ciphertexts via subkeys.

First improved key recovery on AES (Bogdanov, Khovratovich, Rechberger 2011):

AES key length	Exhaustive search	Biclique (expected)
128	2^{128}	$2^{126.1}$
192	2^{192}	$2^{189.7}$
256	2^{256}	$2^{254.4}$

These and other attacks (e.g. square attack) are successful on 8 and lower round AES.

Stream Ciphers

In contrast to block ciphers, stream ciphers don't treat incoming characters independently.

- Encryption C_i of plaintext character M_i depends on internal state of device.
- After encryption, the device changes state according to some rule.

Result: two occurrences of the same plaintext character will usually not result in the same ciphertext character.

Synchronous Stream Ciphers

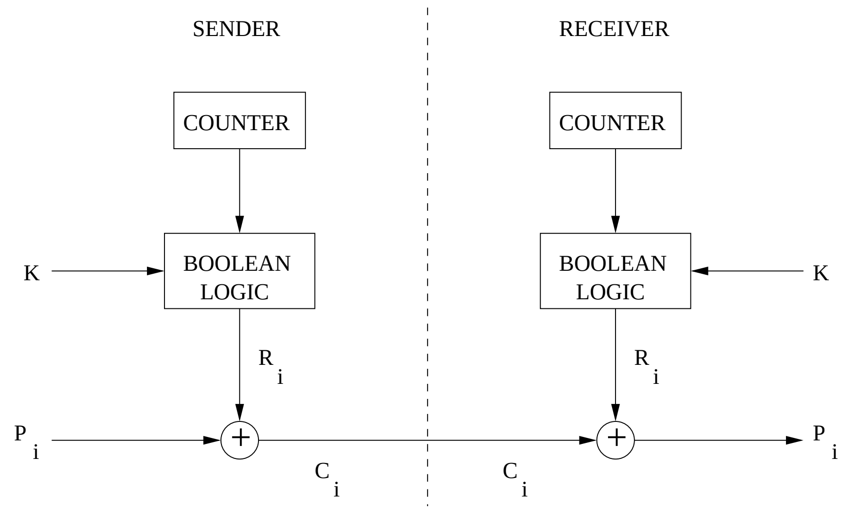
Idea:

- State depends only on the previous state, not on the input M_i.
- C_i depends only on M_i and i, not on $M_{i-1}, M_{i-2}, \dots$
- Implemented by boolean logic that should produce a pseudo-random sequence R_i synchronized by the key (*e.g.* a shift register).

Example 10

The one-time pad can be interpreted as an SSC.

Diagram of an SSC



Block Ciphers as SSCs

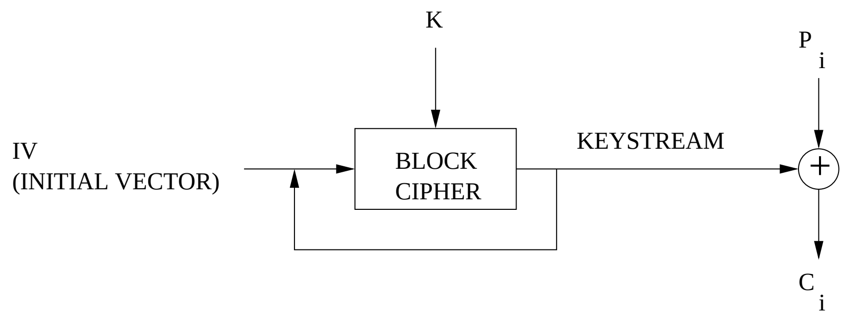
Idea:

- Send an initial key value $KS_0 = IV$ to the receiver in the clear.
- Compute $KS_i = E_K(KS_{i-1})$ and $C_i = M_i \oplus KS_i$.

Problems:

1. No error propagation
2. Loss of one character between sender and receiver destroys synchronization (no memory)

Example: Block-Cipher-based SSC

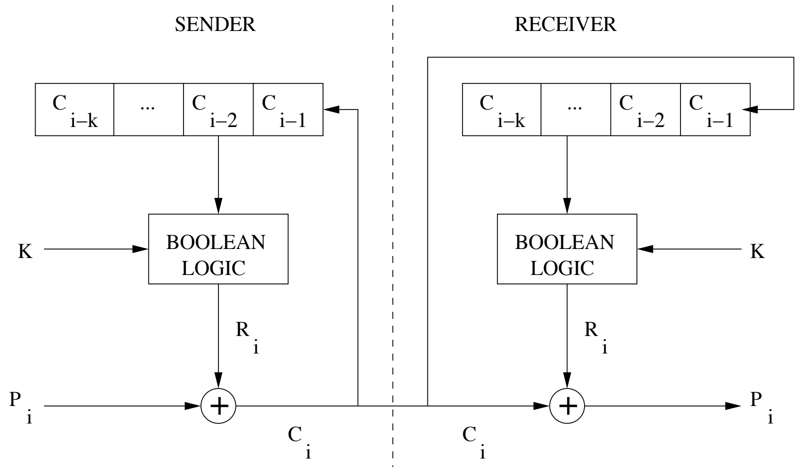


Self-Synchronizing Stream Ciphers (Self-SSC)

Idea:

- Similar to SSC, except the counter is replaced by a register containing the previous k ciphertexts.
- Self-synchronizing after k steps.
- Can also be implemented with a block cipher as above.
- Limited error propagation (k steps).

Diagram of a Self-SSC



ECB Mode

Definition 11 (Electronic code book (ECB) mode)

Blocks are encrypted sequentially, one at a time: $C_i = E_K(M_i)$, $i = 1, 2, \dots$

A block cipher used in ECB mode is essentially a substitution cipher (with all its weaknesses).

Other Modes of Operation

To eliminate the shortcomings of ECB mode, additional modes of operation have been devised:

- Cipher Block Chaining (CBC)
- Output Feedback (OFB)
- Cipher Feedback (CFB)
- Counter (CTR)

DES Certified Modes: ECB, CBC, and CFB; standardized as part of DES standardization process.

- CTR mode arose from concerns with CBC; standardized for AES.

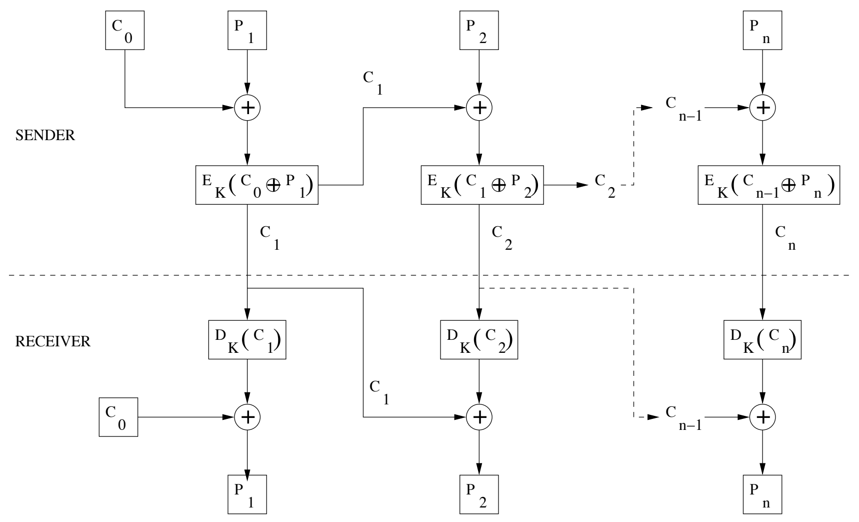
Cipher Block Chaining (CBC) Mode

Send initial *random* block $C_0 = IV$ (e.g. a simple plaintext encrypted in ECB mode, such as $C_0 = E_K(00\cdots000)$

Encryption: $C_i = E_K(\underbrace{M_i \oplus C_{i-1}}_{\text{“Pre-Whitening”}}) \quad i = 1, 2, \ldots$

Decryption: $M_i = D_K(C_i) \oplus C_{i-1} \quad i = 1, 2, \ldots$

Diagram of CBC



Features of CBC

1. Varying *IV* encrypts the same message differently.
2. Repeated plaintexts will be encrypted differently in different repetitions.
3. Plaintext errors propogate through the rest of encryption (good for message authentication, as last ciphertext block depends on all plaintext blocks)
4. Limited error propagation in decryption: error from incorrect ciphertext modification in propagates only to the next block.

Widely used, but vulnerabilities have been discovered (eg. Vaudenay 2002 padding attack, SSL insertion attack).

Counter (CTR) Mode

A counter (CTR_i) of the same size as the cipher block size is maintained.

- Subsequent values of the counter are computed via an iterating function — the FIPS recommendation is simply $CTR_{i+1} = CTR_i + 1 \bmod 2^n$ assuming an n-bit counter.

Encryption: $C_i = E_K(CTR_i) \oplus M_i$

Decryption: $M_i = E_K(CTR_i) \oplus C_i$

Properties of CTR Mode

Counter must be unique for each plaintext block that is ever encrypted under a given key, across all messages.

- can count # of plaintext blocks encrypted under a given counter sequence — new key before exceeding 2^n blocks (n-bit blocks)

Advantages:

- only the encryption function of the block cipher is used (important for AES, in which decryption is slightly less efficient than encryption),
- the ith ciphertext block does not depend on previous ciphertext or plaintext blocks
  - allows for random-access encryption/decryption, parallelism.

Feedback Modes

The feedback modes turn a block cipher into a stream cipher

CFB (cipher feedback) mode is a self-SSC.

- Usually r cipher bits are fed back (for DES, $r = 8$ and IV is at least 48 random bits, right-justified, padded with 0's).
- Each cryptographic session requires a different IV, but these may be sent in the clear.

OFB (output feedback) is a SSC, used similarly to CFB.

Further Information

For more modes of operations as well as recommendations for other block ciphers, see the NIST Crypto Toolkit Modes of Operation page `http://csrc.nist.gov/CryptoToolkit/modes/`.