15-853: Algorithms in the Real World

Cryptography 1 and 2
Cryptography Outline

Introduction: terminology, cryptanalysis, security
Primitives: one-way functions, trapdoors, ...
Protocols: digital signatures, key exchange, ..
Number Theory: groups, fields, ...
Private-Key Algorithms: Rijndael, DES
Public-Key Algorithms: Knapsack, RSA, El-Gamal, ...
Case Studies: Kerberos, SSL
Cryptography Outline

Introduction:
- terminology
- cryptanalytic attacks
- security

Primitives: one-way functions, trapdoors, ...

Protocols: digital signatures, key exchange, ..

Number Theory: groups, fields, ...

Private-Key Algorithms: Rijndael, DES

Public-Key Algorithms: Knapsack, RSA, El-Gamal, ...

Case Studies: Kerberos, SSL
Enigma Machine

"It was thanks to Ultra that we won the war.”
- Winston Churchill
Some Terminology

**Cryptography** - the general term

**Cryptology** - the mathematics

**Encryption** - encoding but sometimes used as general term

**Cryptanalysis** - breaking codes

**Steganography** - hiding message

**Cipher** - a method or algorithm for encrypting or decrypting
More Definitions

Plaintext

Key₁ → Encryption \( E_k(M) = C \)

Cyphertext

Key₂ → Decryption \( D_k(C) = M \)

Original Plaintext

Private Key or Symmetric: Key₁ = Key₂
Public Key or Asymmetric: Key₁ ≠ Key₂
Key₁ or Key₂ is public depending on the protocol
Cryptanalytic Attacks

\( C = \) ciphertext messages  
\( M = \) plaintext messages

**Ciphertext Only:** Attacker has multiple \( C \)s but does not know the corresponding \( M \)s

**Known Plaintext:** Attacker knows some number of \((C,M)\) pairs.

**Chosen Plaintext:** Attacker gets to choose \( M \) and generate \( C \).

**Chosen Ciphertext:** Attacker gets to choose \( C \) and generate \( M \).
What does it mean to be secure?

Unconditionally Secure: Encrypted message cannot be decoded without the key.

Shannon showed in 1943 that key must be as long as the message to be unconditionally secure - this is based on information theory.

A one time pad - xor a random key with a message (Used in 2\textsuperscript{nd} world war).

Security based on computational cost: it is computationally "infeasible" to decode a message without the key.

No (probabilistic) polynomial time algorithm can decode the message.
The Cast

Alice – initiates a message or protocol
Bob – second participant
Trent – trusted middleman
Eve – eavesdropper
Mallory – malicious active attacker
Cryptography Outline

Introduction: terminology, cryptanalysis, security

Primitives:
- one-way functions
- one-way trapdoor functions
- one-way hash functions

Protocols: digital signatures, key exchange, ..

Number Theory: groups, fields, ...

Private-Key Algorithms: Rijndael, DES

Public-Key Algorithms: Knapsack, RSA, El-Gamal, ...

Case Studies: Kerberos, Digital Cash
Primitives: One-Way Functions

(Informally): A function

\[ Y = f(x) \]

is **one-way** if it is easy to compute \( y \) from \( x \) but

“hard” to compute \( x \) from \( y \)

Building block of most cryptographic protocols
And, the security of most protocols rely on their existence.

Unfortunately, not known to exist. This is true even if we assume \( P \neq NP \).
One-way functions: possible definition

1. $F(x)$ is polynomial time
2. $F^{-1}(x)$ is NP-hard

What is wrong with this definition?
One-way functions: better definition

For most $y$ no single PPT (probabilistic polynomial time) algorithm can compute $x$

Roughly: at most a fraction $1/|x|^k$ instances $x$ are easy for any $k$ and as $|x| \to \infty$

This definition can be used to make the probability of hitting an easy instance arbitrarily small.

There are nice results on “pumping” to increase $k$. 
One-way functions: better definition

Also important that cannot get any information about $y = F^{-1}(x)$. Even getting 1 bit of $y$, or some slightly skewed probability over $y$ could be dangerous in an encryption scheme.
Some examples (conjectures)

Factoring:
\[ x = (u,v) \]
\[ y = f(u,v) = u \cdot v \]
If \( u \) and \( v \) are prime it is hard to generate them from \( y \).

Discrete Log: \( y = g^x \mod p \)

where \( p \) is prime and \( g \) is a “generator” (i.e., \( g^1, g^2, g^3, \ldots \) generates all values \(< p\)).

DES with fixed message: \( y = \text{DES}_x(m) \)

This would assume a family of DES functions of increasing key size (for asymptotics)
One-way functions in private-key protocols

\[ y = \text{ciphertext} \quad m = \text{plaintext} \quad k = \text{key} \]

Consider

\[ y = E_k(m) = E(k,m) = E_m(k) \quad (\text{i.e. } f = E_m) \]

should this be a one-way function?

In a **known-plaintext attack** we know a \((y,m)\) pair.
The \(m\) along with \(E\) defines \(f\)

- \(E_m(k)\) needs to be easy
- \(E_m^{-1}(y)\) should be hard

Otherwise we could extract the key \(k\).
One-Way Trapdoor Functions

A one-way function with a “trapdoor”

The trapdoor is a key that makes it easy to invert
the function $y = f(x)$

Example: RSA (conjecture)

$y = x^e \mod n$

Where $n = pq$ ($p$, $q$, $e$ are prime)

$p$ or $q$ or $d$ (where $ed = (p-1)(q-1) \mod n$) can be
used as trapdoors

In public-key algorithms

$f(x) = $ public key (e.g., $e$ and $n$ in RSA)

Trapdoor = private key (e.g., $d$ in RSA)
One-way Hash Functions

\[ Y = h(x) \text{ where} \]

- \( y \) is a fixed length independent of the size of \( x \). In general this means \( h \) is not invertible since it is many to one.

- Calculating \( y \) from \( x \) is easy

- Calculating any \( x \) such that \( y = h(x) \) give \( y \) is hard

Used in digital signatures and other protocols.
Cryptography Outline

Introduction: terminology, cryptanalysis, security
Primitives: one-way functions, trapdoors, ...
Protocols:
  - digital signatures
  - key exchange
Number Theory: groups, fields, ...
Private-Key Algorithms: Rijndael, DES
Public-Key Algorithms: Knapsack, RSA, El-Gamal, ...
Case Studies: Kerberos, Digital Cash
Protocols

Other protocols:
- Authentication
- Secret sharing
- Timestamping services
- Zero-knowledge proofs
- Blind-signatures
- Key-escrow
- Secure elections
- Digital cash

Implementation of the protocol is often the weakest point in a security system.
Protocols: Digital Signatures

Goals:

1. Convince recipient that message was actually sent by a trusted source
2. Do not allow repudiation, \textit{i.e.}, that’s not my signature.
3. Do not allow tampering with the message without invalidating the signature

Item 2 turns out to be really hard to do
Using private keys

- $ka$ is a secret key shared by Alice and Trent
- $kb$ is a secret key shared by Bob and Trent

$\text{sig}$ is a note from Trent saying that Alice “signed” it. To prevent repudiation, Trent needs to keep $m$ or at least $h(m)$ in a database.
Using Public Keys

\[ D_{k_1}(m) \]

Alice \rightarrow Bob

K1 = Alice’s private key
Bob decrypts it with her public key

More Efficiently

\[ D_{k_1}(h(m)) + m \]

Alice \rightarrow Bob

h(m) is a one-way hash of m
Key Exchange

Private Key method

Trent

Generates $k$

Alice

$E_{ka}(k)$

Bob

$E_{kb}(k)$

Public Key method

Alice

Evaluates $E_{k1}(k)$

Bob

$k1 = Bob's public key$

Key exchange protocol (e.g. Diffie-Hellman)
Cryptography Outline

Introduction: terminology, cryptanalysis, security
Primitives: one-way functions, trapdoors, ...
Protocols: digital signatures, key exchange, ..
Number Theory Review: (Mostly covered last week)
  - Groups
  - Fields
  - Polynomials and Galois fields
Private-Key Algorithms: Rijndael, DES
Public-Key Algorithms: Knapsack, RSA, El-Gamal, ...
Case Studies: Kerberos, Digital Cash
Number Theory Outline

Groups
- Definitions, Examples, Properties
- Multiplicative group modulo n
- The Euler-phi function

Fields
- Definition, Examples
- Polynomials
- Galois Fields

Why does number theory play such an important role?

It is the mathematics of finite sets of values.
Groups

A Group \((G, *, I)\) is a set \(G\) with operator \(*\) such that:

1. **Closure.** For all \(a, b \in G\), \(a * b \in G\)

2. **Associativity.** For all \(a, b, c \in G\), \(a*(b*c) = (a*b)*c\)

3. **Identity.** There exists \(I \in G\), such that for all \(a \in G\), \(a*I = I*a = a\)

4. **Inverse.** For every \(a \in G\), there exist a unique element \(b \in G\), such that \(a*b = b*a = I\)

An **Abelian or Commutative Group** is a Group with the additional condition

5. **Commutativity.** For all \(a, b \in G\), \(a*b = b*a\)
Examples of groups

- Integers, Reals or Rationals with Addition
- The nonzero Reals or Rationals with Multiplication
- Non-singular $n \times n$ real matrices with Matrix Multiplication
- Permutations over $n$ elements with composition

$$[0\rightarrow 1, 1\rightarrow 2, 2\rightarrow 0] \circ [0\rightarrow 1, 1\rightarrow 0, 2\rightarrow 2] = [0\rightarrow 0, 1\rightarrow 2, 2\rightarrow 1]$$

We will only be concerned with finite groups, I.e., ones with a finite number of elements.
Key properties of finite groups

Notation: \( a^j \equiv a \ast a \ast a \ast \ldots \ast \) \( j \) times

Theorem (Fermat’s little): for any finite group \((G,*,I)\) and \( g \in G, g^{\vert G \vert} = I \)

Definition: the order of \( g \in G \) is the smallest positive integer \( m \) such that \( g^m = I \)

Definition: a group \( G \) is cyclic if there is a \( g \in G \) such that \( \text{order}(g) = \vert G \vert \)

Definition: an element \( g \in G \) of order \( \vert G \vert \) is called a generator or primitive element of \( G \).
Groups based on modular arithmetic

The group of positive integers modulo a prime $p$

$\mathbb{Z}_p^* \equiv \{1, 2, 3, ..., p-1\}$

$\ast_p \equiv \text{multiplication modulo } p$

Denoted as: $(\mathbb{Z}_p^*, \ast_p)$

Required properties

3. Identity. 1.
4. Inverse. Yes.

Example: $\mathbb{Z}_7^* = \{1, 2, 3, 4, 5, 6\}$

$1^{-1} = 1, \ 2^{-1} = 4, \ 3^{-1} = 5, \ 6^{-1} = 6$
Other properties

\[ |\mathbb{Z}_p^*| = (p-1) \]
By Fermat’s little theorem: \( a^{(p-1)} = 1 \pmod{p} \)

Example of \( \mathbb{Z}_7^* \)

<table>
<thead>
<tr>
<th></th>
<th>( x )</th>
<th>( x^2 )</th>
<th>( x^3 )</th>
<th>( x^4 )</th>
<th>( x^5 )</th>
<th>( x^6 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>6</td>
<td>4</td>
<td>5</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>1</td>
<td>4</td>
<td>2</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>6</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>6</td>
<td>1</td>
<td>6</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

Generators

For all \( p \) the group is cyclic.
What if n is not a prime?

The group of positive integers modulo a non-prime n
\[ Z_n = \{1, 2, 3, \ldots, n-1\}, \text{ n not prime} \]
* \[ p \equiv \text{multiplication modulo n} \]

Required properties?
1. Closure. ?
2. Associativity. ?
3. Identity. ?
4. Inverse. ?

How do we fix this?
Groups based on modular arithmetic

The multiplicative group modulo \( n \)

\[ Z_n^* \equiv \{ m : 1 \leq m < n, \gcd(n, m) = 1 \} \]

\( * \equiv \) multiplication modulo \( n \)

Denoted as \( (Z_n^*, *) \)

Required properties:

- Closure. Yes.
- Associativity. Yes.
- Identity. 1.
- Inverse. Yes.

Example: \( Z_{15}^* = \{ 1, 2, 4, 7, 8, 11, 13, 14 \} \)

\( 1^{-1} = 1, \ 2^{-1} = 8, \ 4^{-1} = 4, \ 7^{-1} = 13, \ 11^{-1} = 11, \ 14^{-1} = 14 \)
The Euler Phi Function

\[ \phi(n) = \left| \mathbb{Z}_n^* \right| = n \prod_{p|n} (1 - 1/p) \]

If \( n \) is a product of two primes \( p \) and \( q \), then

\[ \phi(n) = pq(1 - 1/p)(1 - 1/q) = (p - 1)(q - 1) \]

Note that by Fermat’s Little Theorem:

\[ a^{\phi(n)} = 1 \pmod{n} \quad \text{for} \quad a \in \mathbb{Z}_n^* \]

Or for \( n = pq \)

\[ a^{(p-1)(q-1)} = 1 \pmod{n} \quad \text{for} \quad a \in \mathbb{Z}_{pq}^* \]

This will be very important in RSA!
Generators

Example of \( Z_{10}^* \): \{1, 3, 7, 9\}

<table>
<thead>
<tr>
<th></th>
<th>( x )</th>
<th>( x^2 )</th>
<th>( x^3 )</th>
<th>( x^4 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>9</td>
<td>7</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>9</td>
<td>3</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>1</td>
<td>9</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

For \( n = (2, 4, p^e, 2p^e) \), \( p \) an odd prime, \( Z_n \) is cyclic.
Operations we will need

**Multiplication:** $a \times b \pmod{n}$
- Can be done in $O(\log^2 n)$ bit operations, or better

**Power:** $a^k \pmod{n}$
- The power method $O(\log n)$ steps, $O(\log^3 n)$ bit ops

```python
def pow(a, k):
    if (k = 0):
        return 1
    else if (k mod 2 = 1):
        return a * (pow(a, k/2))
    else:
        return (pow(a, k/2))^2
```

**Inverse:** $a^{-1} \pmod{n}$
- Euclid's algorithm $O(\log n)$ steps, $O(\log^3 n)$ bit ops
**Euclid’s Algorithm**

**Euclid’s Algorithm:**
\[
gcd(a,b) = gcd(b, a \mod b)
\]
\[
gcd(a,0) = a
\]

“Extended” Euclid’s algorithm:
- Find \( x \) and \( y \) such that \( ax + by = gcd(a,b) \)
- Can be calculated as a side-effect of Euclid’s algorithm.
- Note that \( x \) and \( y \) can be zero or negative.

This allows us to find \( a^{-1} \mod n \), for \( a \in Z_n^* \)

In particular return \( x \) in \( ax + ny = 1 \).
Euclid's Algorithm

fun euclid(a,b) =
    if (b = 0) then a
    else euclid(b, a mod b)

fun ext_euclid(a,b) =
    if (b = 0) then (a, 1, 0)
    else
        let (d, x, y) = ext_euclid(b, a mod b)
        in (d, y, x - (a/b) y)
    end

The code is in the form of an inductive proof.

Exercise: prove the inductive step
Discrete Logarithms

If $g$ is a generator of $\mathbb{Z}_n^*$, then for all $y$ there is a unique $x \pmod{\phi(n)}$ such that

- $y = g^x \mod n$

This is called the **discrete logarithm** of $y$ and we use the notation

- $x = \log_g(y)$

In general finding the discrete logarithm is conjectured to be hard. It is as hard as factoring.
Polynomials with coefficients in GF($p^n$)

We can make a finite field by using an irreducible polynomial $M(x)$ selected from $GF(p^n)[x]$.

For an order $m$ polynomial and by abuse of notation we write: $GF(GF(p^n)^m)$, which has $p^{nm}$ elements.

Used in Reed-Solomon codes and Rijndael.

- In Rijndael $p=2$, $n=8$, $m=4$, i.e. each coefficient is a byte, and each element is a 4 byte word (32 bits).

Note: all finite fields are isomorphic to $GF(p^n)$, so this is really just another representation of $GF(2^{32})$. This representation, however, has practical advantages.
Cryptography Outline

Introduction: terminology, cryptanalysis, security
Primitives: one-way functions, trapdoors, ...
Protocols: digital signatures, key exchange, ..
Number Theory: groups, fields, ...
Private-Key Algorithms:
  - Block ciphers and product ciphers
  - Rijndael, DES
  - Cryptanalysis
Public-Key Algorithms: Knapsack, RSA, El-Gamal, ...
Case Studies: Kerberos, Digital Cash
Private Key Algorithms

Plaintext

Key₁ → Encryption \( E_k(M) = C \)

Cyphertext

Key₁ → Decryption \( D_k(C) = M \)

Original Plaintext

What granularity of the message does \( E_k \) encrypt?
**Private Key Algorithms**

**Block Ciphers**: blocks of bits at a time
- DES (Data Encryption Standard)
  - Banks, linux passwords (almost), TSL, kerberos, ...
- Blowfish
- IDEA (used in PGP, TSL as option)
- Rijdael (AES) - the current standard

**Stream Ciphers**: one bit (or a few bits) at a time
- RC4 (TSL as option)
- PKZip
- Sober, Leviathan, Panama, ...
Private Key: Block Ciphers

Encrypt one block at a time (e.g. 64 bits)

\[ c_i = f(k,m_i) \quad m_i = f'(k,c_i) \]

Keys and blocks are often about the same size.

Equal message blocks will encrypt to equal codeblocks

- Why is this a problem?

Various ways to avoid this:

- E.g. \( c_i = f(k,c_{i-1} \text{ xor } m_i) \)
  
  “Cipher block chaining” (CBC)

Why could this still be a problem?

**Solution**: attach random block to the front of the message
Securit of block ciphers

Ideal:
- k-bit -> k-bit key-dependent subsitution (i.e. “random permutation”)
- If keys and blocks are k-bits, can be implemented with $2^{2k}$ entry table as a key!!!!!!

Completely impractical.
Iterated Block Ciphers

Consists of $n$ rounds

$R = \text{the "round" function}$

$s_i = \text{state after round } i$

$k_i = \text{the } i^{th} \text{ round key}$
Iterated Block Ciphers: Decryption

Run the rounds in reverse. Requires that $R$ has an inverse.

\[
\begin{align*}
R^{-1} & \quad \text{key} \\
R^{-1} & \quad k_1 \\
R^{-1} & \quad k_2 \\
R^{-1} & \quad k_n \\
c & \quad s_2 \\
s_1 & \\
& \quad \ldots
\end{align*}
\]
Feistel Networks

If function is not invertible rounds can still be made invertible. Requires 2 rounds to mix all bits.

Forwards

Backwards

Used by DES (the Data Encryption Standard)
Product Ciphers

Each round has two components:

- **Substitution** on smaller blocks
  Decorrelate input and output: “confusion”

- **Permutation** across the smaller blocks
  Mix the bits: “diffusion”

**Substitution-Permutation Product Cipher**

**Avalanche Effect**: 1 bit of input should affect all output bits, ideally evenly, and for all settings of other input bits
Rijndael (AES)

Selected by AES (Advanced Encryption Standard, part of NIST) as the new private-key encryption standard in 2002 over DES.

Based on an open “competition”.
- Narrowed to 5 Sept. 1999
  - MARS by IBM, RC6 by RSA, Twofish by Counterplane, Serpent, and Rijndael
- Official May 2002 (AES page on Rijndael)

Designed by Rijmen and Daemen (Dutch)
Goals of Rijndael

Resistance against known attacks:
- Differential cryptanalysis
- Linear cryptanalysis
- Truncated differentials
- Square attacks
- Interpolation attacks
- Weak and related keys

Speed + Memory efficiency across platforms
- 32-bit processors
- 8-bit processors (e.g. smart cards)
- Dedicated hardware

Design simplicity and clearly stated security goals
High-level overview

An iterated block cipher with
- 10-14 rounds,
- 128-256 bit blocks, and
- 128-256 bit keys

Mathematically reasonably sophisticated
Blocks and Keys

The blocks and keys are organized as matrices of bytes. For the 128-bit case, it is a 4x4 matrix.

\[
\begin{pmatrix}
    b_0 & b_4 & b_8 & b_{12} \\
    b_1 & b_5 & b_9 & b_{13} \\
    b_2 & b_6 & b_{10} & b_{14} \\
    b_3 & b_7 & b_{11} & b_{15}
\end{pmatrix}
\begin{pmatrix}
    k_0 & k_4 & k_8 & k_{12} \\
    k_1 & k_5 & k_9 & k_{13} \\
    k_2 & k_6 & k_{10} & k_{14} \\
    k_3 & k_7 & k_{11} & k_{15}
\end{pmatrix}
\]

Data block  Key

\(b_0, b_1, ..., b_{15}\) is the order of the bytes in the stream.
Galois Fields in Rijndael

Uses $GF(2^8)$ over bytes.
The irreducible polynomial is:
\[ M(x) = x^8 + x^4 + x^3 + x + 1 \text{ or } 100011011 \text{ or } 0x11B \]

Also uses degree 3 polynomials with coefficients from $GF(2^8)$.
These are kept as 4 bytes (used for the columns)
The polynomial used as a modulus is:
\[ M(x) = 00000001x^4 + 00000001 \text{ or } x^4 + 1 \]
Not irreducible, but we only need to find inverses of polynomials that are relatively prime to it.
Each round

The inverse runs the steps and rounds backwards. Each step must be reversible!
Byte Substitution

Non linear: \( y = b^{-1} \) (done over \( GF(2^8) \))
Linear: \( z = Ay + B \) (done over \( GF(2) \), i.e., binary)

\[
A = \begin{pmatrix}
1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\
1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\
1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\
1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\
\vdots
\end{pmatrix}
\quad B = \begin{pmatrix}
1 \\
1 \\
0 \\
0 \\
0 \\
1 \\
1 \\
0
\end{pmatrix}
\]

To invert the substitution:
\( y = A^{-1}(z - B) \) (the matrix \( A \) is nonsingular)
\( b = y^{-1} \) (over \( GF(2^8) \))
**Mix Columns**

For each column $a$ in data block

\[
\begin{align*}
&a_0 \\
&a_1 \\
&a_2 \\
&a_3
\end{align*}
\]

compute $b(x) = (a_3x^3 + a_2x^2 + a_1x + a_0)(3x^3 + x^2 + x + 2) \mod x^4 + 1$

where coefficients are taken over $GF(2^8)$.

New column $b$ is

\[
\begin{align*}
&b_0 \\
&b_1 \\
&b_2 \\
&b_3
\end{align*}
\]

where $b(x) = b_3x^3 + b_2x^2 + b_1x + b_0$
Implementation

Using \(x^j \mod (x^4 + 1) = x^{(j \mod 4)}\)

\((a_3x^3+a_2x^2+a_1x+a_0)(3x^3+x^2+x+2) \mod x^4+1\)

\[= (2a_0+3a_1+a_2+a_3) +\]
\[ (a_0+2a_1+3a_2+a_3)x +\]
\[ (a_0+a_1+2a_2+3a_3)x^2 +\]
\[ (3a_0+a_1+a_2+2a_3)x^3 \]

Therefore, \(b = C \cdot a\)

\[C = \begin{pmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{pmatrix}\]

\(M(x)\) is not irreducible, but the rows of \(C\) and \(M(x)\) are coprime, so the transform can be inverted.
Generating the round keys

Words corresponding to columns of the key

\[ f = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{bmatrix} \rightarrow \begin{bmatrix} b_2 \\ b_3 \\ b_4 \\ b_1 \end{bmatrix} \rightarrow \begin{bmatrix} \text{rotate} \\ \text{sub byte} \\ \text{const}_i \end{bmatrix} \]
Performance

Performance: (64-bit AMD Athlon 2.2Ghz, 2005, Open SSL):

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Bits/key</th>
<th>Mbits/sec</th>
</tr>
</thead>
<tbody>
<tr>
<td>DES-cbc</td>
<td>56</td>
<td>399</td>
</tr>
<tr>
<td>Blowfish-cbc</td>
<td>128</td>
<td>703</td>
</tr>
<tr>
<td>Rijndael-cbc</td>
<td>128</td>
<td>917</td>
</tr>
</tbody>
</table>

Intel X86 now has AES instructions (since 2008)
With instructions Intel-i7 gives 12Gbits/sec/core
# X86 Instructions

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AESENC</td>
<td>Perform one round of an AES encryption flow</td>
</tr>
<tr>
<td>AESENCLAST</td>
<td>Perform the last round of an AES encryption flow</td>
</tr>
<tr>
<td>AESDEC</td>
<td>Perform one round of an AES decryption flow</td>
</tr>
<tr>
<td>AESDECLAST</td>
<td>Perform the last round of an AES decryption flow</td>
</tr>
<tr>
<td>AESKEYGENASSIST</td>
<td>Assist in AES round key generation</td>
</tr>
<tr>
<td>AESIMC</td>
<td>Assist in AES Inverse Mix Columns</td>
</tr>
<tr>
<td>PCLMULQDQ</td>
<td>Carryless multiply ([CLMUL][3])</td>
</tr>
</tbody>
</table>
Linear Cryptanalysis

A known plaintext attack used to extract the key

Consider a linear equality involving $i$, $o$, and $k$
- e.g.: $k_1 + k_6 = i_2 + i_4 + i_5 + o_4$
To be secure this should be true with $p = .5$
   (probability over all inputs and keys)
If true with $p = 1$, then linear and easy to break
If true with $p = .5 + \varepsilon$ then you might be able to use
   this to help break the system
Differential Cryptanalysis

A chosen plaintext attack used to extract the key

\[ I \rightarrow K \rightarrow \text{Round} \rightarrow O \]

Considers fixed “differences” between inputs, \( \Delta_I = I_1 - I_2 \), and sees how they propagate into differences in the outputs, \( \Delta_O = O_1 - O_2 \).

“difference” is often exclusive OR

Assigns probabilities to different keys based on these differences. With enough and appropriate samples \((I_1, I_2, O_1, O_2)\), the probability of a particular key will converge to 1.