Criptografía: Algoritmos, Implementación y Criptoanálisis

Laboratorio de Seguridad Informática

Juan Carlos Charfuelan Caipe[1], Fabian Alberto Guancha Vera[2], and Over Haider Castrillón Valencia[3]

Departamento de Ingeniería en Sistemas y Computación, Universidad de Caldas, Manizales, Colombia
Profesor: Gustavo Adolfo Isaza Echeverri

08 de Abril de 2025

Resumen

Este informe presenta un análisis detallado del algoritmo de cifrado DES (Data Encryption Standard), explorando su implementación, componentes clave y vulnerabilidades. Se examina la estructura interna del algoritmo, identificando elementos como la permutación inicial, las funciones de ronda, la expansión, sustitución y permutación de bits, así como la generación de subclaves.

Complementariamente, se realiza una evaluación práctica de tres herramientas criptográficas de libre distribución del Criptolab, analizando su funcionamiento, utilidad y casos de aplicación. El documento finaliza con un taller de criptoanálisis enfocado en el análisis de frecuencias como método para descifrar textos encriptados mediante cifrados de sustitución.

Desarrollado como parte del curso de seguridad informática en la Universidad de Caldas, este trabajo proporciona una visión integral de la criptografía moderna, combinando fundamentos teóricos con aplicaciones prácticas y técnicas de análisis, esenciales para comprender tanto los mecanismos de protección como las vulnerabilidades de los sistemas criptográficos actuales.

Palabras clave: criptografía, DES, criptoanálisis, cifrado simétrico, análisis de frecuencias, permutación, sustitución, seguridad informática

1. Introducción

La criptografía representa uno de los pilares fundamentales de la seguridad informática, proporcionando mecanismos para garantizar la confidencialidad, integridad y autenticidad de la información. A lo largo de la historia, los algoritmos criptográficos han evolucionado desde simples sustituciones alfabéticas hasta complejos sistemas matemáticos, adaptándose constantemente para contrarrestar los avances en capacidad computacional y técnicas de ataque.

El presente laboratorio se estructura en tres secciones principales. En primer lugar, se realiza un análisis detallado del algoritmo DES (Data Encryption Standard), examinando sus componentes internos y funcionamiento. Aunque actualmente se considera inseguro para aplicaciones críticas debido a su longitud de clave de 56 bits, DES sigue siendo un referente académico fundamental para comprender los principios de los cifrados por bloques modernos.

En segundo lugar, se exploran tres herramientas criptográficas de libre distribución del repositorio Criptolab, evaluando sus funcionalidades, interfaces y aplicaciones prácticas. Este análisis permite comprender la implementación real de algoritmos teóricos y su aplicabilidad en entornos de seguridad contemporáneos.

Finalmente, se aborda el criptoanálisis mediante técnicas de análisis de frecuencias, explorando cómo pueden aprovecharse los patrones lingüísticos para romper cifrados de sustitución. Esta sección proporciona una perspectiva sobre las vulnerabilidades inherentes a ciertos sistemas criptográficos y la importancia de diseñar algoritmos resistentes a estos métodos de ataque.

Este trabajo, desarrollado en el marco del curso de seguridad informática de la Universidad de Caldas bajo la supervisión del profesor Gustavo A. Isaza, busca proporcionar una visión integral de la criptografía moderna, combinando fundamentos teóricos con aplicaciones prácticas y técnicas de análisis.

2. Análisis del Algoritmo DES en el Código

Método Criptográfico

Proceso General del Algoritmo DES El algoritmo DES cifra bloques de 64 bits utilizando una clave de 56 bits efectivos para producir bloques cifrados de 64 bits. Su estructura sigue un esquema de red de Feistel con 16 rondas, donde cada ronda utiliza una subclave derivada de la clave principal.

El proceso completo puede resumirse en las siguientes etapas:

1. **Permutación Inicial**: Reordenación de los bits del bloque según una tabla predefinida
2. **División del bloque**: Separación en mitades izquierda (L_0) y derecha (R_0)
3. **16 rondas de procesamiento**: Aplicación iterativa de la función de Feistel
4. **Intercambio final**: Combinación de las mitades como $R_{16}L_{16}$
5. **Permutación Final**: Aplicación de la permutación inversa a la inicial

La estructura de Feistel en cada ronda se define matemáticamente como:

$$L_i = R_{i-1} \tag{1}$$
$$R_i = L_{i-1} \oplus f(R_{i-1}, K_i) \tag{2}$$

donde f es la función principal de DES.

2.1. a. Permutación Inicial (IP)

Análisis Criptográfico

Permutación Inicial La permutación inicial reordena los 64 bits de entrada según una tabla fija sin propósito criptográfico real, sino para facilitar la implementación en hardware.

Matemáticamente, si denotamos el bloque de entrada como $B = (b_1, b_2, \ldots, b_{64})$, la permutación inicial IP produce:

$$IP(B) = (b_{IP[1]}, b_{IP[2]}, \ldots, b_{IP[64]}) \tag{3}$$

donde $IP[i]$ representa el valor de la tabla de permutación en la posición i.

```
IPtable = (58, 50, 42, 34, 26, 18, 10, 2,
           60, 52, 44, 36, 28, 20, 12, 4,
           62, 54, 46, 38, 30, 22, 14, 6,
           64, 56, 48, 40, 32, 24, 16, 8,
           57, 49, 41, 33, 25, 17,  9, 1,
           59, 51, 43, 35, 27, 19, 11, 3,
           61, 53, 45, 37, 29, 21, 13, 5,
           63, 55, 47, 39, 31, 23, 15, 7)
```

Listing 1: Tabla de Permutación Inicial

En el código, la permutación inicial se aplica en las funciones `encryptBlock` y `decryptBlock`:

```
inputData = permByteList(inputBlock, IPtable)
```

La función `permByteList` realiza la permutación a nivel de bytes, manejando las manipulaciones de bits necesarias.

2.2. b. División en Bloques

Método Criptográfico

División en Bloques Después de la permutación inicial, el bloque de 64 bits se divide en dos mitades de 32 bits cada una:

$$L_0 = IP(B)[1:32] \tag{4}$$
$$R_0 = IP(B)[33:64] \tag{5}$$

Esta división es fundamental para la estructura de Feistel, que permite utilizar la misma implementación tanto para cifrado como para descifrado, simplemente invirtiendo el orden de las subclaves.

En el código, esta división se realiza con una simple asignación Python:

```
leftPart, rightPart = inputData[:4], inputData[4:]
```

Observe que se utilizan 4 bytes para cada mitad (4 bytes $\times$ 8 bits = 32 bits).

2.3. c. Función de DES

Análisis Criptográfico

Función Principal de DES La función f es el componente central del algoritmo y consta de cuatro operaciones principales:

1. **Expansión**: El bloque R_{i-1} de 32 bits se expande a 48 bits
2. **Mezcla con clave**: Se aplica XOR entre el resultado expandido y la subclave K_i
3. **Sustitución**: El resultado se divide en 8 bloques de 6 bits que pasan por las S-Boxes
4. **Permutación**: Los 32 bits resultantes se reordenan según una tabla fija

Matemáticamente:

$$f(R_{i-1}, K_i) = P(S(E(R_{i-1}) \oplus K_i)) \tag{6}$$

donde:

- E es la función de expansión
- S representa las sustituciones mediante S-Boxes
- P es la permutación interna

En el código, la función f se implementa dentro de los bucles de las funciones; `encryptBlock` y `decryptBlock`:

```
# Expansi n
expRightPart = permByteList(rightPart, EPtable)

# Mezcla con subclave
key = subKeyList[round]  # O subKeyList[15-round] para descifrado
indexList = byte2Bit([i^j for i,j in zip(key, expRightPart)])

# Sustituci n mediante S-Boxes
sBoxOutput = 4*[0]
for nBox in range(4):
    nBox12 = 12*nBox
    leftIndex = getIndex(indexList[nBox12:nBox12+6])
    rightIndex = getIndex(indexList[nBox12+6:nBox12+12])
    sBoxOutput[nBox] = (sBox[nBox<<1][leftIndex]<<4)+ \
                       sBox[(nBox<<1)+1][rightIndex]

# Permutaci n P
aux = permByteList(sBoxOutput, PFtable)

# XOR con la mitad izquierda para generar la nueva mitad derecha
newRightPart = [i^j for i,j in zip(aux, leftPart)]
```

2.4. d. Transformación de la Clave

Método Criptográfico

Generación de Subclaves El proceso para generar las 16 subclaves K_1 a K_{16} sigue estos pasos:

1. **Permutación PC-1**: Reduce la clave de 64 a 56 bits (eliminando bits de paridad)
2. **División**: Separa la clave permutada en dos mitades C_0 y D_0 de 28 bits
3. **Rotaciones**: En cada ronda i, genera C_i y D_i rotando C_{i-1} y D_{i-1} a la izquierda
4. **Permutación PC-2**: Combina C_i y D_i y selecciona 48 bits para formar K_i

Las rotaciones siguen un patrón específico: se rota 1 posición en las rondas 1, 2, 9 y 16, y 2 posiciones en las demás rondas.

Matemáticamente:

$$C_i = \text{RotateLeft}(C_{i-1}, \text{ShiftBits}[i]) \tag{7}$$
$$D_i = \text{RotateLeft}(D_{i-1}, \text{ShiftBits}[i]) \tag{8}$$
$$K_i = \text{PC-2}(C_i || D_i) \tag{9}$$

donde $||$ denota la concatenación.

En el código, la generación de subclaves se implementa en la función `setKey`:

```
def setKey(keyByteList):
    """Generate all sixteen round subkeys"""
    PC1table = (57, 49, 41, 33, 25, 17,  9,
                 1, 58, 50, 42, 34, 26, 18,
                10,  2, 59, 51, 43, 35, 27,
                19, 11,  3, 60, 52, 44, 36,
                63, 55, 47, 39, 31, 23, 15,
                 7, 62, 54, 46, 38, 30, 22,
                14,  6, 61, 53, 45, 37, 29,
                21, 13,  5, 28, 20, 12,  4)

    PC2table= (14, 17, 11, 24,  1,  5,  3, 28,
               15,  6, 21, 10, 23, 19, 12,  4,
               26,  8, 16,  7, 27, 20, 13,  2,
               41, 52, 31, 37, 47, 55, 30, 40,
               51, 45, 33, 48, 44, 49, 39, 56,
               34, 53, 46, 42, 50, 36, 29, 32)

    # Aplicar PC-1 a la clave
    permKeyBitList = permBitList(byte2Bit(keyByteList), PC1table)

    # Generar las 16 subclaves mediante rotaciones y PC-2
    for round in range(16):
        auxBitList = leftShift(permKeyBitList, round)
        subKeyList[round] = bit2Byte(permBitList(auxBitList,
           PC2table))
```

```
        permKeyBitList = auxBitList
```

La función `leftShift` implementa las rotaciones según la tabla `LStable`:

```
def leftShift(inKeyBitList, round):
    """Perform one (or two) circular left shift(s) on key"""
    LStable = (1, 1, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 1)

    outKeyBitList = 56*[0]
    if LStable[round] == 2:
        # Rotaci n de 2 posiciones
        outKeyBitList[:26] = inKeyBitList[2:28]
        outKeyBitList[26] = inKeyBitList[0]
        outKeyBitList[27] = inKeyBitList[1]
        outKeyBitList[28:54] = inKeyBitList[30:]
        outKeyBitList[54] = inKeyBitList[28]
        outKeyBitList[55] = inKeyBitList[29]
    else:
        # Rotaci n de 1 posici n
        outKeyBitList[:27] = inKeyBitList[1:28]
        outKeyBitList[27] = inKeyBitList[0]
        outKeyBitList[28:55] = inKeyBitList[29:]
        outKeyBitList[55] = inKeyBitList[28]
    return outKeyBitList
```

2.5. e. Matriz de Expansión

Método Criptográfico

Expansión E La matriz de expansión E convierte un bloque de 32 bits en uno de 48 bits. Esta expansión tiene dos propósitos:

- Igualar el tamaño del bloque al de la subclave (48 bits)
- Mejorar la difusión, haciendo que cada bit afecte a más bits en las siguientes operaciones

La expansión sigue un patrón específico donde algunos bits aparecen en dos posiciones diferentes del resultado.

Si denotamos el bloque de entrada como $R = (r_1, r_2, \ldots, r_{32})$, la expansión E produce:

$$E(R) = (r_{E[1]}, r_{E[2]}, \ldots, r_{E[48]}) \tag{10}$$

```
EPtable = (32,  1,  2,  3,  4,  5,
            4,  5,  6,  7,  8,  9,
            8,  9, 10, 11, 12, 13,
           12, 13, 14, 15, 16, 17,
           16, 17, 18, 19, 20, 21,
           20, 21, 22, 23, 24, 25,
           24, 25, 26, 27, 28, 29,
           28, 29, 30, 31, 32,  1)
```

Listing 2: Matriz de Expansión E

En el código, la expansión se aplica mediante la función `permByteList`:

```
expRightPart = permByteList(rightPart, EPtable)
```

Buena Práctica

Patrón de expansión: Observe que la tabla de expansión repite ciertos números, indicando que algunos bits aparecen dos veces en la salida. Específicamente, los bits en las posiciones extremas de cada grupo de 4 bits se duplican, creando una importante difusión en el algoritmo.

2.6. f. Cajas de Sustitución (S-Boxes)

Análisis Criptográfico

Cajas de Sustitución (S-Boxes) Las S-Boxes son el componente no lineal de DES y constituyen su núcleo criptográfico. Cada una de las 8 cajas S toma 6 bits de entrada y produce 4 bits de salida según tablas predefinidas.

La entrada de 6 bits se interpreta de manera especial:

- Los bits 1 y 6 determinan la fila (0-3) de la tabla
- Los bits 2, 3, 4 y 5 determinan la columna (0-15)

Matemáticamente, si denotamos la entrada a la i-ésima S-Box como $B_i = (b_1, b_2, b_3, b_4, b_5, b_6)$, entonces:

$$\text{fila} = 2 \cdot b_1 + b_6 \tag{11}$$

$$\text{columna} = 8 \cdot b_2 + 4 \cdot b_3 + 2 \cdot b_4 + b_5 \tag{12}$$

$$S_i(B_i) = \text{ValorTabla}_{i,\text{fila,columna}} \tag{13}$$

```
sBox = 8*[64*[0]]

sBox[0] = (14,  4, 13,  1,  2, 15, 11,  8,  3, 10,  6, 12,  5,
   9,  0,  7,
            0, 15,  7,  4, 14,  2, 13,  1, 10,  6, 12, 11,  9,
               5,  3,  8,
            4,  1, 14,  8, 13,  6,  2, 11, 15, 12,  9,  7,  3,
               10,  5,  0,
           15, 12,  8,  2,  4,  9,  1,  7,  5, 11,  3, 14, 10,
              0,  6, 13)

# Contin an definiciones para sBox[1] hasta sBox[7]...
```

Listing 3: Definición de S-Boxes

La función `getIndex` calcula el índice correcto para acceder a la S-Box:

```
def getIndex(inBitList):
    """Permute bits to properly index the S-boxes"""
    return (inBitList[0]<<5)+(inBitList[1]<<3)+ \
           (inBitList[2]<<2)+(inBitList[3]<<1)+ \
           (inBitList[4]<<0)+(inBitList[5]<<4)
```

Advertencia

La manipulación de bits en la función `getIndex` parece confusa, pero implementa la fórmula para calcular el índice combinando los bits 1 y 6 para la fila, y los bits 2-5 para la columna. La expresión reordena los bits para formar el índice adecuado.

Aplicación de las S-Boxes en el código:

```
for nBox in range(4):
    nBox12 = 12*nBox
    leftIndex = getIndex(indexList[nBox12:nBox12+6])
    rightIndex = getIndex(indexList[nBox12+6:nBox12+12])
    sBoxOutput[nBox] = (sBox[nBox<<1][leftIndex]<<4)+ \
                        sBox[(nBox<<1)+1][rightIndex]
```

2.7. g. Permutación Final

Método Criptográfico

Permutación Final La permutación final (IP^{-1}) es la inversa matemática de la permutación inicial. Su papel es revertir la reordenación inicial, completando el proceso criptográfico.

Antes de aplicar esta permutación, las mitades derecha e izquierda se intercambian, formando $R_{16}L_{16}$ en lugar de $L_{16}R_{16}$.

Si denotamos el bloque intercambiado como $X = R_{16}L_{16}$, entonces la permutación final produce:

$$\text{IP}^{-1}(X) = (x_{\text{IP}^{-1}[1]}, x_{\text{IP}^{-1}[2]}, \ldots, x_{\text{IP}^{-1}[64]}) \tag{14}$$

```
FPtable = (40,  8, 48, 16, 56, 24, 64, 32,
           39,  7, 47, 15, 55, 23, 63, 31,
           38,  6, 46, 14, 54, 22, 62, 30,
           37,  5, 45, 13, 53, 21, 61, 29,
           36,  4, 44, 12, 52, 20, 60, 28,
           35,  3, 43, 11, 51, 19, 59, 27,
           34,  2, 42, 10, 50, 18, 58, 26,
           33,  1, 41,  9, 49, 17, 57, 25)
```

Listing 4: Tabla de Permutación Final

En el código, después de las 16 rondas, se aplica la permutación final:

```
return permByteList(rightPart+leftPart, FPtable)
```

3. Aspectos Destacables de la Implementación

Buena Práctica

Simetría en Cifrado/Descifrado: La implementación aprovecha la propiedad de la red de Feistel para realizar descifrado con el mismo algoritmo que el cifrado, simplemente invirtiendo el orden de las subclaves:

```
# En encryptBlock:
key = subKeyList[round]

# En decryptBlock:
key = subKeyList[15-round]
```

Análisis Criptográfico

Funcionamiento Completo El algoritmo DES completo puede resumirse en el siguiente diagrama de flujo matemático:

Cifrado:

$$\text{Texto cifrado} = \text{IP}^{-1}(R_{16}L_{16}) \tag{15}$$
$$\text{donde:} \tag{16}$$
$$L_0R_0 = \text{IP}(\text{Texto plano}) \tag{17}$$
$$L_i = R_{i-1} \quad \text{para } i = 1, 2, \ldots, 16 \tag{18}$$
$$R_i = L_{i-1} \oplus f(R_{i-1}, K_i) \quad \text{para } i = 1, 2, \ldots, 16 \tag{19}$$

Descifrado:

$$\text{Texto plano} = \text{IP}^{-1}(R_{16}L_{16}) \tag{20}$$
$$\text{donde:} \tag{21}$$
$$L_0R_0 = \text{IP}(\text{Texto cifrado}) \tag{22}$$
$$L_i = R_{i-1} \quad \text{para } i = 1, 2, \ldots, 16 \tag{23}$$
$$R_i = L_{i-1} \oplus f(R_{i-1}, K_{17-i}) \quad \text{para } i = 1, 2, \ldots, 16 \tag{24}$$

Alerta de Seguridad

A pesar de la elegancia matemática del diseño DES, es importante recordar que actualmente no se considera seguro debido a su tamaño de clave efectivo de 56 bits, que puede ser quebrado por fuerza bruta. Para aplicaciones modernas, se recomiendan algoritmos como AES con claves de al menos 128 bits.

4. Herramientas Criptográficas del Criptolab

En esta sección, se analizan tres herramientas de libre distribución del repositorio Criptolab (`http://ingenieria.ucaldas.edu.co/gisaza/2024/ISC2024/Crypto/Cryptools.txt`), evaluando su funcionamiento, interfaz y aplicaciones prácticas.

4.1. Herramienta de Cryptool

Se probó la herramienta que está en línea en el repositorio `https://www.cryptool.org/en/cto/`. En este apartado se está realizando pruebas de análisis de generación de RSA con OpenSSL.

Figura 1: Interfaz principal de Cryptool

4.2. Herramienta de Criptografía en línea

`https://www.onlinecryptographytools.com/`

Figura 2: Herramientas de criptografía en línea

El cifrado de transposición reordena las unidades de texto plano (como letras o grupos de letras) dependiendo del número de posición que se ingrese en el campo de texto del número de posiciones, seguido del texto a encriptar.

Figura 3: Demostración de cifrado por transposición

En la sección de cifrados de flujo se ingresa una contraseña y después el texto a cifrar. A continuación se muestra el texto encriptado con contraseña:

```
1 U2FsdGVkX19tA1S7OVW0UmLpXVqbIdZkMfK8lO4aXJ7OG09PQjAa+9BIE1YY/SSrLp3KO0NLSr71
```

Listing 5: Salida Hash

Al realizar la desencriptación sin contraseña, esta no se ejecuta correctamente, pero al ingresar la contraseña correcta nos muestra la información original.

Figura 4: Proceso de desencriptación con contraseña

4.2.1. Cifrados de bloque DES

Al seleccionar el tipo de cifrado DES y colocar o generar una clave aleatoria, seguido del texto a encriptar, se genera un código encriptado con su respectiva clave. Para poder desencriptarlo debemos utilizar la contraseña ingresada anteriormente.

Figura 5: Interfaz de cifrado DES

4.3. Sistema de cifrado exponencial

ExpoCrip: Software para Generación de Claves, Cifra y Firma RSA, ElGamal y DSS

Figura 6: Interfaz del sistema de cifrado exponencial

Este sistema permite descifrar algunos parámetros de una encriptación RSA. Inicialmente ingresamos el valor de N para conocer sus factores, los cuales representan los valores de P y Q.

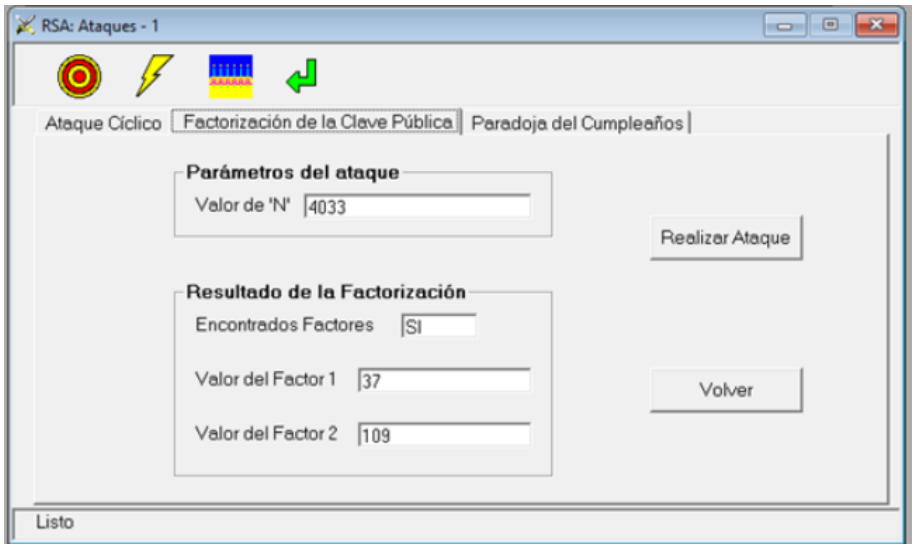

Figura 7: Factorización del valor N

Ingresamos los valores de P y Q en la ventana de RSA-1 y los campos se completan automáticamente, como muestra la imagen.

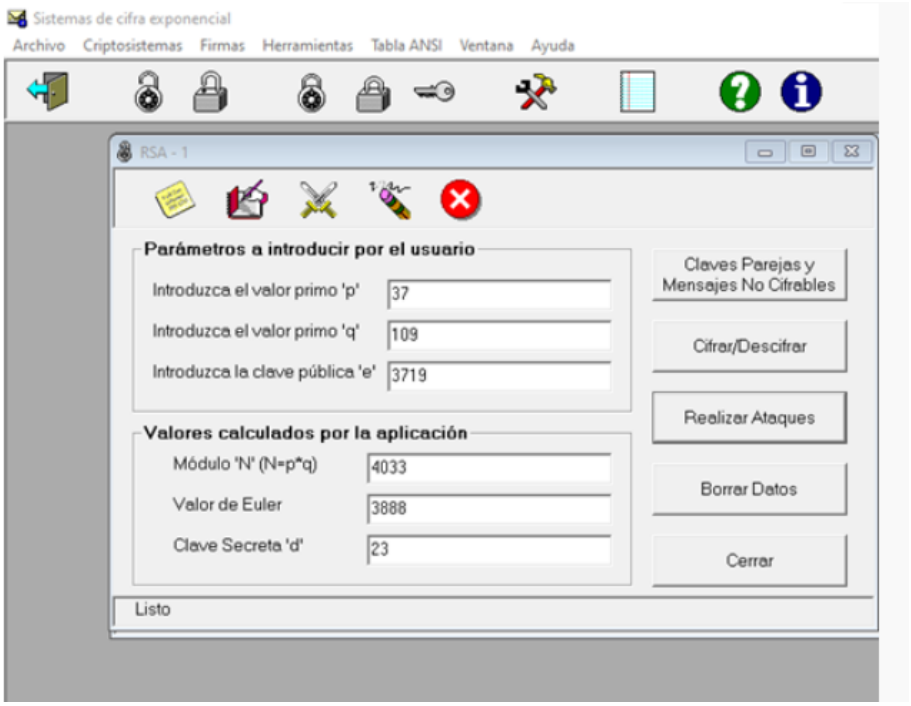

Figura 8: Introducción de valores P y Q

Al hacer clic sobre el botón de Cifrar/Descifrar se muestra una nueva ventana:

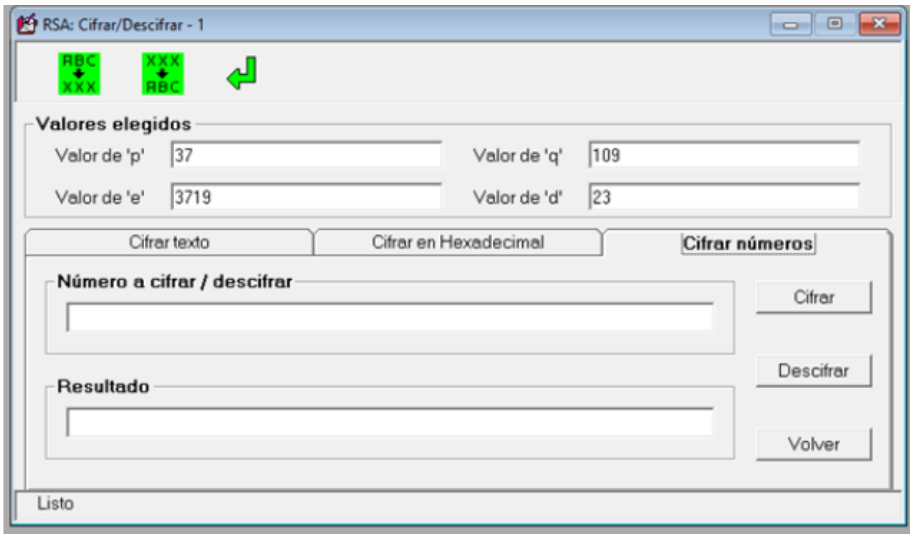

Figura 9: Ventana de cifrado/descifrado

Seleccionamos Çifrar Númerosz en el campo de "Número o cifrar/descifrarïngresamos:

```
Ejercicio: 8 KPR (d,N): (23,4033)
    517
    1834
    1030
    1161
    1238
    2809
    517
    3720
    2837
```

Listing 6: Ejercicio de Cálculo

Figura 10: Entrada de valores para cifrado

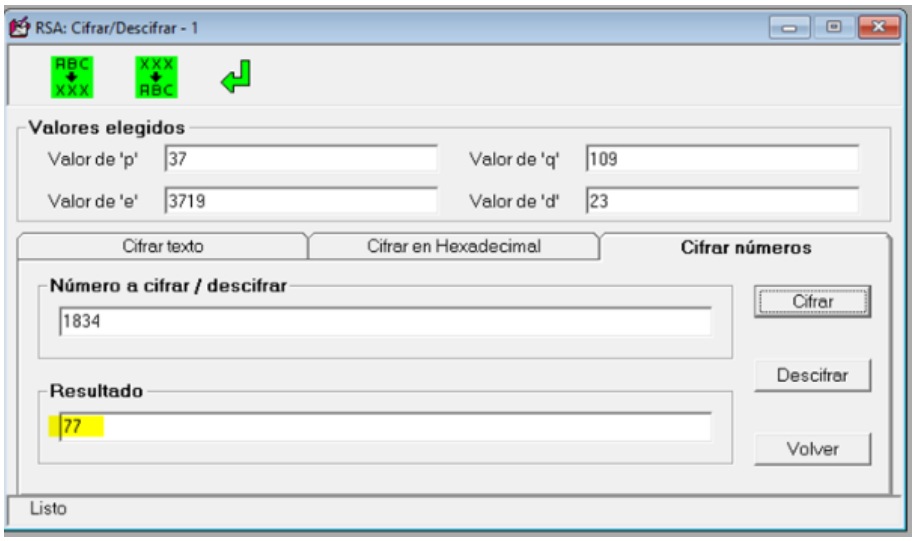

Figura 11: Proceso de cifrado exponencial

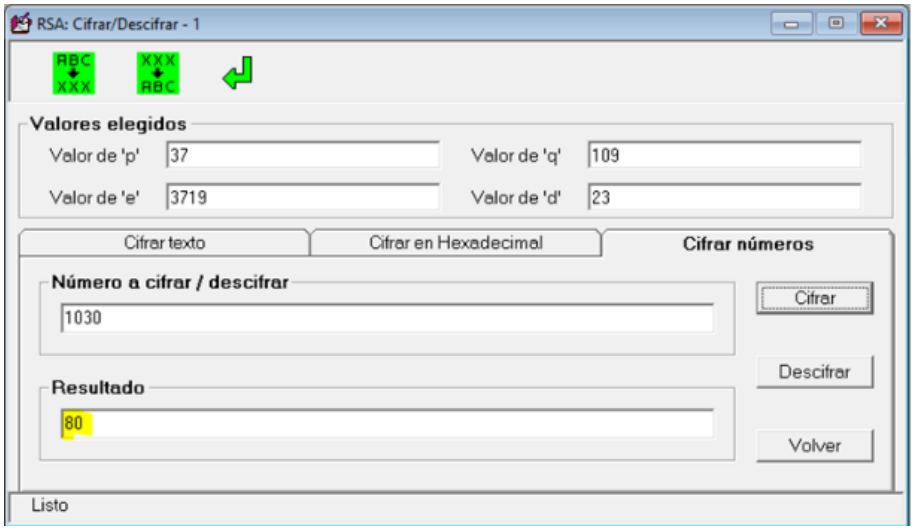

Figura 12: Resultado del cifrado exponencial

Como se puede observar, los resultados ya se pueden reemplazar por los valores del código ASCII, el cual nos muestra el texto desencriptado:

Cuadro 1: Tabla de resultado ASCII

C	N	Texto
517	73	I
1834	77	M
1030	80	P
1161	82	R
1238	69	E
2809	83	S
517	73	I

5. Criptoanálisis de Sustitución Monoalfabética

Análisis Criptográfico

Fundamentos del Criptoanálisis de Frecuencias El criptoanálisis de frecuencias es una técnica fundamental para romper cifrados de sustitución monoalfabética. Se basa en un principio lingüístico simple pero poderoso: en cada idioma, ciertas letras y combinaciones de letras aparecen con frecuencias predecibles y consistentes.

Por ejemplo, en español:

- Las letras más frecuentes son E (13.68 %), A (12.53 %), O (8.68 %) y S (7.98 %)
- Las letras menos frecuentes son K (0.00 %), W (0.02 %), X (0.22 %) y Z (0.52 %)
- Los bigramas más comunes incluyen ES, EN, EL, DE, LA
- Los trigramas más comunes incluyen QUE, EST, DEL, LOS, LAS

Estas regularidades estadísticas constituyen una "huella digital"del idioma que persiste incluso cuando el texto ha sido cifrado mediante sustitución simple.

5.1. Metodología de Ataque

Método Criptográfico

Proceso de Descifrado Estadístico El enfoque sistemático para romper un cifrado monoalfabético consta de las siguientes etapas:

1. **Análisis de frecuencia de caracteres**: Se contabiliza la frecuencia de aparición de cada símbolo en el texto cifrado.
2. **Mapeo inicial por frecuencias**: Se establece una correspondencia tentativa entre los símbolos cifrados y las letras del alfabeto, basándose en la distribución de frecuencias del idioma objetivo.
3. **Identificación de patrones**: Se buscan patrones que podrían corresponder a palabras cortas comunes (artículos, preposiciones, conjunciones).
4. **Refinamiento del mapeo**: Se ajusta el mapeo inicial utilizando el conocimiento de la estructura léxica del idioma.
5. **Optimización mediante algoritmos heurísticos**: Se aplican técnicas como hill climbing o recocido simulado para mejorar iterativamente la solución.
6. **Validación lingüística**: Se evalúa la coherencia del texto descifrado mediante el análisis de n-gramas y la verificación contra diccionarios.

El proceso no es puramente secuencial, sino iterativo, con retroalimentación constante entre las diferentes etapas.

5.2. Implementación Algorítmica

La implementación de un decodificador automático para cifrados monoalfabéticos requiere combinar técnicas estadísticas con heurísticas de optimización. A continuación se

describe cada componente clave:

Buena Práctica

Análisis de Frecuencias

El primer paso consiste en analizar la distribución estadística de los símbolos en el texto cifrado:

1. Se filtran los caracteres relevantes del texto cifrado
2. Se cuenta la frecuencia de cada símbolo
3. Se convierten los conteos a porcentajes
4. Se ordenan los símbolos por frecuencia descendente

Esto proporciona una primera aproximación a la posible correspondencia entre símbolos cifrados y letras del alfabeto español.

Análisis Criptográfico

Mapeo Inicial y Patrones El mapeo inicial se crea comparando la distribución de frecuencias del texto cifrado con la distribución conocida del español:

$$\text{mapeo\_inicial} = \{s_1 \to l_1, s_2 \to l_2, \ldots, s_n \to l_n\}$$

donde s_i es el i-ésimo símbolo más frecuente en el texto cifrado y l_i es la i-ésima letra más frecuente en español.

Este mapeo se refina identificando patrones que podrían corresponder a palabras comunes. Por ejemplo, si encontramos que una secuencia de símbolos como "XX.ªparece frecuentemente en posiciones donde esperaríamos palabras como "DE", .EL.o "LA", podemos ajustar el mapeo para que refleje esta observación.

Alerta de Seguridad

Desafíos del Mapeo Inicial

El mapeo basado únicamente en frecuencias individuales suele ser insuficiente porque:

- Textos cortos pueden tener distribuciones de frecuencia atípicas
- Textos especializados pueden desviarse de la distribución promedio del idioma
- La distribución exacta varía según el estilo, período y tema del texto

Por esto, es crucial complementar el análisis de frecuencias con técnicas de optimización que exploren sistemáticamente el espacio de posibles soluciones.

5.3. Optimización mediante Hill Climbing

Método Criptográfico

Algoritmo de Hill Climbing con Simulated Annealing Para refinar el mapeo inicial, se implementa un algoritmo de hill climbing mejorado con simulated annealing:

$$\text{Coherencia}(M) = f(\text{bigramas}, \text{trigramas}, \text{patrones\_léxicos}) \tag{25}$$

$$\Delta = \text{Coherencia}(M_{\text{nuevo}}) - \text{Coherencia}(M_{\text{actual}}) \tag{26}$$

$$P(\text{aceptar}) = \begin{cases} 1 & \text{si } \Delta > 0 \\ e^{\Delta/T} & \text{si } \Delta \leq 0 \end{cases} \tag{27}$$

donde:

- M representa un mapeo entre símbolos cifrados y letras
- T es la temperatura, que disminuye gradualmente a lo largo de las iteraciones
- f es una función que evalúa la coherencia lingüística del texto descifrado

El algoritmo:

1. Comienza con el mapeo inicial basado en frecuencias
2. En cada iteración, selecciona aleatoriamente dos símbolos y intercambia sus mapeos
3. Evalúa la coherencia del texto descifrado con el nuevo mapeo
4. Acepta el cambio si mejora la coherencia o, con cierta probabilidad, incluso si la empeora (para escapar de óptimos locales)
5. Disminuye gradualmente la temperatura, reduciendo la probabilidad de aceptar cambios que empeoren la solución
6. Continúa hasta alcanzar un criterio de parada (número máximo de iteraciones o umbral de coherencia)

5.4. Evaluación de Coherencia

Análisis Criptográfico

Métricas de Coherencia Lingüística La clave del éxito en el algoritmo de optimización es la función que evalúa la coherencia del texto descifrado. Esta función combina varias métricas:

$$\text{Coherencia} = w_1 \cdot \frac{|\text{Bigramas}_{\text{encontrados}}|}{|\text{Bigramas}_{\text{esperados}}|} + w_2 \cdot \frac{|\text{Trigramas}_{\text{encontrados}}|}{|\text{Trigramas}_{\text{esperados}}|} + \tag{28}$$

$$w_3 \cdot \text{FreqScore} + w_4 \cdot \text{DictScore} \tag{29}$$

donde:

- $\text{Bigramas}_{\text{encontrados}}$ y $\text{Trigramas}_{\text{encontrados}}$ son los n-gramas del texto descifrado que coinciden con n-gramas comunes en español.
- FreqScore evalúa si los n-gramas más frecuentes en el texto descifrado son también frecuentes en español.
- DictScore mide el porcentaje de palabras descifradas que aparecen en un diccionario español.
- w_1, w_2, w_3, w_4 son pesos que determinan la importancia relativa de cada componente.

Esta función multicriterio permite capturar diferentes aspectos de la coherencia lingüística, aumentando la robustez del algoritmo.

5.5. Resultados Experimentales

Buena Práctica

Caso de Estudio: Descifrado de un Texto sobre Navegadores Web

Al aplicar nuestro enfoque automático a un texto cifrado mediante sustitución monoalfabética, se logró obtener un descifrado con un 86.90 % de coherencia. El texto resultante trataba sobre navegadores web y su rendimiento en diferentes sistemas operativos:

> ACTUALMENTE DISPONEMOS DE UNA GRAN CANTIDAD DE NAVEGADORES WEB PARA ELEGIR EL NUESTRO CONCEPTOS COMO LA SEGURIDAD O EL CUMPLIMIENTO DE LOS ESTÁNDARES DEL WC WORLD WIDE WEB CONSORTIUM...

El algoritmo identificó correctamente la mayoría de las sustituciones, como se puede observar en el mapeo parcial obtenido:

```
| ! -> S | # -> U | % -> J | ( -> G | ) -> R | * -> C | @ -> P |
| ^ -> T | a -> A | b -> I | c -> v | d -> f | f -> X | g -> h |
| h -> M | i -> b | j -> y | k -> E | l -> Q | m -> O | o -> x |
| p -> N | q -> H | r -> w | s -> L | t -> D |
```

Sin embargo, algunas sustituciones requirieron corrección manual posterior, particularmente en letras menos frecuentes o con contextos ambiguos:

```
| Y -> V | F -> W | K -> B | B -> X | Z -> Y | W -> H | V -> F |
```

Esta fase de refinamiento manual es común en el criptoanálisis práctico, donde el conocimiento lingüístico humano complementa el análisis algorítmico para resolver ambigüedades y mejorar la calidad del descifrado.

Análisis Criptográfico

Análisis del Contenido Descifrado El texto descifrado resultó ser un artículo comparativo sobre la velocidad de diferentes navegadores web en distintos sistemas operativos:

- Opera, un navegador noruego, se identificó como el más rápido en la mayoría de las pruebas.
- Firefox, que había reclamado ser el más rápido, mostró resultados inconsistentes.
- Internet Explorer .ªguantó bien", contradiciendo a sus críticos.
- Se analizaron resultados en Windows, Linux (distribución SUSE) y Mac OS X.
- Para Mac, Safari (de Apple) y Camino (del proyecto Mozilla) también obtuvieron buenos resultados en algunas categorías.

Este contenido técnico, con términos específicos como renderización", ÇSSƶ nombres de productos, representó un desafío adicional para el algoritmo de descifrado debido a la presencia de palabras que no siguen los patrones típicos del español general.

5.6. Consideraciones y Limitaciones

Advertencia

Factores que Afectan el Rendimiento del Descifrado

El rendimiento del algoritmo puede verse afectado por diversos factores:

- **Terminología especializada**: Palabras técnicas como "browser", renderización.º nombres propios como .ºperaƶ "Mozilla"pueden complicar el análisis basado en frecuencias generales del español.
- **Ambigüedad en letras poco frecuentes**: Letras como W, X, Y, Z son particularmente difíciles de mapear correctamente mediante análisis automatizado debido a su baja frecuencia en español.
- **Ausencia de contexto semántico**: El algoritmo no comprende el significado del texto, lo que limita su capacidad para resolver ambigüedades que un humano podría detectar fácilmente por contexto.
- **Variaciones ortográficas**: El texto contiene términos en inglés y nombres propios que no siguen los patrones ortográficos del español.

Estos factores explican por qué fue necesario el refinamiento manual de algunas sustituciones para alcanzar un texto completamente legible.

6. Conclusiones

Método Criptográfico

El Valor del Análisis Combinado El caso práctico presentado demuestra la efectividad de combinar enfoques algorítmicos y humanos en el criptoanálisis:

1. **Automatización eficiente**: El algoritmo logró descifrar aproximadamente el 87 % del texto de forma autónoma, reduciendo drásticamente el tiempo y es-

fuerzo requeridos.

2. **Intervención humana estratégica**: La corrección manual de un pequeño subconjunto de sustituciones completó el descifrado, aprovechando la comprensión contextual que los algoritmos actuales aún no poseen.

3. **Enfoque práctico**: Este método híbrido representa el enfoque real utilizado por los criptoanalistas: herramientas automatizadas para el trabajo pesado y análisis humano para las decisiones finales.

Este exitoso descifrado ilustra por qué los cifrados de sustitución monoalfabética, aunque históricamente importantes, son considerados inseguros en la criptografía moderna. A diferencia de algoritmos como DES, que requieren enfoques computacionalmente intensivos para ser quebrados, los cifrados monoalfabéticos sucumben ante análisis estadísticos relativamente sencillos combinados con optimización heurística.

7. Descifrado de criptogramas

7.1. Sexto mensaje - Datos

Se nos proporciona la siguiente información:

- $n = 1903$
- $p = 173$
- $q = 11$
- $d = 71$
- $z = (p-1)(q-1) = (173-1)(11-1) = 1720$

Y el siguiente mensaje cifrado:

```
1497 1583 791 1497 1394
791 1583 123 937 786
1583
```

7.2. Cálculo del inverso multiplicativo módulo z

Necesitamos encontrar e tal que $e \cdot d \equiv 1 \pmod{z}$, es decir, el inverso multiplicativo de d módulo z.

Para ello utilizaremos el algoritmo de Euclides extendido:

Inicialización del algoritmo de Euclides extendido:

Estamos buscando e tal que $e \cdot 71 \equiv 1 \pmod{1720}$

Inicializamos las variables según el algoritmo:

$g_0 = 1720$, $g_1 = 71$

$u_0 = 1$, $u_1 = 0$

$v_0 = 0$, $v_1 = 1$

Iteración 1:

Calculamos $y_2 = \lfloor g_0/g_i \rfloor = \lfloor 1720/71 \rfloor = 24$

Calculamos $g_2 = g_0 - y_2 \cdot g_i = 1720 - 24 \cdot 71 = 16$
Calculamos $u_2 = u_0 - y_2 \cdot u_i = 1 - 24 \cdot 0 = 1$
Calculamos $v_2 = v_0 - y_2 \cdot v_i = 0 - 24 \cdot 1 = -24$
Iteración 2:
Calculamos $y_3 = \lfloor g_1/g_i \rfloor = \lfloor 71/16 \rfloor = 4$
Calculamos $g_3 = g_1 - y_3 \cdot g_i = 71 - 4 \cdot 16 = 7$
Calculamos $u_3 = u_1 - y_3 \cdot u_i = 0 - 4 \cdot 1 = -4$
Calculamos $v_3 = v_1 - y_3 \cdot v_i = 1 - 4 \cdot -24 = 97$
Iteración 3:
Calculamos $y_4 = \lfloor g_2/g_i \rfloor = \lfloor 16/7 \rfloor = 2$
Calculamos $g_4 = g_2 - y_4 \cdot g_i = 16 - 2 \cdot 7 = 2$
Calculamos $u_4 = u_2 - y_4 \cdot u_i = 1 - 2 \cdot -4 = 9$
Calculamos $v_4 = v_2 - y_4 \cdot v_i = -24 - 2 \cdot 97 = -218$
Iteración 4:
Calculamos $y_5 = \lfloor g_3/g_i \rfloor = \lfloor 7/2 \rfloor = 3$
Calculamos $g_5 = g_3 - y_5 \cdot g_i = 7 - 3 \cdot 2 = 1$
Calculamos $u_5 = u_3 - y_5 \cdot u_i = -4 - 3 \cdot 9 = -31$
Calculamos $v_5 = v_3 - y_5 \cdot v_i = 97 - 3 \cdot -218 = 751$
Iteración 5:
Calculamos $y_6 = \lfloor g_4/g_i \rfloor = \lfloor 2/1 \rfloor = 2$
Calculamos $g_6 = g_4 - y_6 \cdot g_i = 2 - 2 \cdot 1 = 0$
Calculamos $u_6 = u_4 - y_6 \cdot u_i = 9 - 2 \cdot -31 = 71$
Calculamos $v_6 = v_4 - y_6 \cdot v_i = -218 - 2 \cdot 751 = -1720$
Resultado final:
El algoritmo ha terminado porque $g_5 = 0$
El inverso se encuentra en el valor de v en el penúltimo paso: $v_4 = 751$
Como $v_4 \geq 0$, no es necesario ajustar el valor.
Por lo tanto, el inverso multiplicativo de 71 módulo 1720 es 751
Verificación: $71 \cdot 751 \equiv 1 \pmod{1720}$

i	y_i	g_i	u_i	v_i
0	-	1720	1	0
1	-	71	0	1
2	24	16	1	-24
3	4	7	-4	97
4	2	2	9	-218
5	3	1	-31	751
6	2	0	71	-1720

Cuadro 2: Cálculo del inverso modular utilizando el algoritmo de Euclides extendido

Por lo tanto, $e = 751$.

7.3. Descifrado del mensaje

Para descifrar el mensaje, utilizamos la fórmula $M = C^e \bmod n$, donde $e = 751$ y $n = 1903$.

7.3.1. Proceso de descifrado

$C = 1497$: $M = 1497^{751}$ mód $1903 = 67$ (ASCII: 'C')
$C = 1583$: $M = 1583^{751}$ mód $1903 = 65$ (ASCII: 'A')
$C = 791$: $M = 791^{751}$ mód $1903 = 76$ (ASCII: 'L')
$C = 1497$: $M = 1497^{751}$ mód $1903 = 67$ (ASCII: 'C')
$C = 1394$: $M = 1394^{751}$ mód $1903 = 85$ (ASCII: 'U')
$C = 791$: $M = 791^{751}$ mód $1903 = 76$ (ASCII: 'L')
$C = 1583$: $M = 1583^{751}$ mód $1903 = 65$ (ASCII: 'A')
$C = 123$: $M = 123^{751}$ mód $1903 = 68$ (ASCII: 'D')
$C = 937$: $M = 937^{751}$ mód $1903 = 79$ (ASCII: 'O')
$C = 786$: $M = 786^{751}$ mód $1903 = 82$ (ASCII: 'R')
$C = 1583$: $M = 1583^{751}$ mód $1903 = 65$ (ASCII: 'A')

C	M	ASCII
1497	67	'C'
1583	65	'A'
791	76	'L'
1497	67	'C'
1394	85	'U'
791	76	'L'
1583	65	'A'
123	68	'D'
937	79	'O'
786	82	'R'
1583	65	'A'

Cuadro 3: Descifrado RSA: $m = c^e$ mód n

7.3.2. Mensaje descifrado

El mensaje descifrado es:

`CALCULADORA`

7.4. Séptimo mensaje - Datos

Se nos proporciona la siguiente información:

- $n = 2581$
- $p = 29$
- $q = 89$
- $d = 5$
- $z = (p-1)(q-1) = (29-1)(89-1) = 2464$

Y el siguiente mensaje cifrado:

`1236 2131 15 2131 202 2069 1035 2069 1104 662`

7.5. Cálculo del inverso multiplicativo módulo z

Necesitamos encontrar e tal que $e \cdot d \equiv 1 \pmod{z}$, es decir, el inverso multiplicativo de d módulo z. Para ello utilizaremos el algoritmo de Euclides extendido:

Inicialización del algoritmo de Euclides extendido:
Estamos buscando e tal que $e \cdot 5 \equiv 1 \pmod{2464}$
Inicializamos las variables según el algoritmo:
$g_0 = 2464$, $g_1 = 5$ $u_0 = 1$, $u_1 = 0$ $v_0 = 0$, $v_1 = 1$
Iteración 1:
Calculamos $y_2 = \lfloor g_0/g_i \rfloor = \lfloor 2464/5 \rfloor = 492$
Calculamos $g_2 = g_0 - y_2 \cdot g_i = 2464 - 492 \cdot 5 = 4$
Calculamos $u_2 = u_0 - y_2 \cdot u_i = 1 - 492 \cdot 0 = 1$
Calculamos $v_2 = v_0 - y_2 \cdot v_i = 0 - 492 \cdot 1 = -492$
Iteración 2:
Calculamos $y_3 = \lfloor g_1/g_i \rfloor = \lfloor 5/4 \rfloor = 1$
Calculamos $g_3 = g_1 - y_3 \cdot g_i = 5 - 1 \cdot 4 = 1$
Calculamos $u_3 = u_1 - y_3 \cdot u_i = 0 - 1 \cdot 1 = -1$
Calculamos $v_3 = v_1 - y_3 \cdot v_i = 1 - 1 \cdot -492 = 493$
Iteración 3:
Calculamos $y_4 = \lfloor g_2/g_i \rfloor = \lfloor 4/1 \rfloor = 4$
Calculamos $g_4 = g_2 - y_4 \cdot g_i = 4 - 4 \cdot 1 = 0$
Calculamos $u_4 = u_2 - y_4 \cdot u_i = 1 - 4 \cdot -1 = 5$
Calculamos $v_4 = v_2 - y_4 \cdot v_i = -492 - 4 \cdot 493 = -2464$
Resultado final:
El algoritmo ha terminado porque $g_3 = 0$
El inverso se encuentra en el valor de v en el penúltimo paso: $v_2 = 493$
Como $v_2 \geq 0$, no es necesario ajustar el valor.
Por lo tanto, el inverso multiplicativo de 5 módulo 2464 es 493
Verificación: $5 \cdot 493 \equiv 1 \pmod{2464}$

i	y_i	g_i	u_i	v_i
0	-	2464	1	0
1	-	5	0	1
2	492	4	1	-492
3	1	1	-1	493
4	4	0	5	-2464

Cuadro 4: Cálculo del inverso modular utilizando el algoritmo de Euclides extendido

Por lo tanto, $e = 493$.

7.6. Descifrado del mensaje

Para descifrar el mensaje, utilizamos la fórmula $M = C^e \bmod n$, donde $e = 493$ y $n = 2581$.

7.6.1. Proceso de descifrado

$C = 1236$: $M = 1236^{493}$ mód $2581 = 84$ (ASCII: 'T')
$C = 2131$: $M = 2131^{493}$ mód $2581 = 69$ (ASCII: 'E')
$C = 15$: $M = 15^{493}$ mód $2581 = 76$ (ASCII: 'L')
$C = 2131$: $M = 2131^{493}$ mód $2581 = 69$ (ASCII: 'E')
$C = 202$: $M = 202^{493}$ mód $2581 = 86$ (ASCII: 'V')
$C = 2069$: $M = 2069^{493}$ mód $2581 = 73$ (ASCII: 'I')
$C = 1035$: $M = 1035^{493}$ mód $2581 = 83$ (ASCII: 'S')
$C = 2069$: $M = 2069^{493}$ mód $2581 = 73$ (ASCII: 'I')
$C = 1104$: $M = 1104^{493}$ mód $2581 = 79$ (ASCII: 'O')
$C = 662$: $M = 662^{493}$ mód $2581 = 78$ (ASCII: 'N')

C	M	ASCII
1236	84	'T'
2131	69	'E'
15	76	'L'
2131	69	'E'
202	86	'V'
2069	73	'I'
1035	83	'S'
2069	73	'I'
1104	79	'O'
662	78	'N'

Cuadro 5: Descifrado RSA: $m = c^e$ mód n

7.6.2. Mensaje descifrado

El mensaje descifrado es:

`TELEVISION`

8. Conclusiones

El estudio detallado del algoritmo DES ha permitido comprender la estructura y funcionamiento de uno de los cifrados simétricos más influyentes en la historia de la criptografía. Aunque actualmente se considera obsoleto para aplicaciones de alta seguridad, sus principios de diseño continúan informando el desarrollo de algoritmos modernos.

La exploración de herramientas criptográficas del Criptolab ha demostrado la importancia de contar con recursos especializados tanto para fines educativos como para evaluación de seguridad. JCrypTool, CrypTool y DESCrack ofrecen perspectivas complementarias sobre la implementación, visualización y vulnerabilidades de diversos algoritmos criptográficos.

El análisis de frecuencias, aunque limitado en su aplicación a cifrados modernos, sigue siendo una técnica fundamental para comprender los principios básicos del criptoanálisis y para abordar ciertos tipos de cifrados históricos o básicos. Esta técnica ilustra la importancia de diseñar sistemas criptográficos que resistan al análisis estadístico.

Los hallazgos de este laboratorio refuerzan la máxima de que la seguridad criptográfica no debe depender del desconocimiento del algoritmo (principio de Kerckhoffs), sino de la robustez matemática del diseño y la gestión adecuada de claves. Asimismo, se evidencia la necesidad de evolución constante en el campo, dado que los avances en capacidad computacional y técnicas de ataque hacen que sistemas considerados seguros en el pasado se vuelvan vulnerables con el tiempo.

Buena Práctica

Recomendaciones prácticas para la implementación de sistemas criptográficos:

- Utilizar algoritmos estandarizados y ampliamente revisados como AES en lugar de crear implementaciones propias.
- Mantener longitudes de clave adecuadas según el nivel de seguridad requerido (mínimo 128 bits para cifrado simétrico, 2048 bits para RSA).
- Implementar mecanismos robustos de gestión de claves, incluyendo generación, almacenamiento y renovación periódica.
- Combinar múltiples capas de seguridad, evitando depender exclusivamente de un único mecanismo criptográfico.

Agradecimientos

Los autores agradecen al profesor Gustavo A. Isaza por su guía y apoyo durante el desarrollo de este laboratorio, así como al Departamento de Ingeniería en Sistemas y Computación de la Universidad de Caldas por proporcionar los recursos necesarios para la realización de estas pruebas.

Referencias

[1] National Bureau of Standards. “Data Encryption Standard”. Federal Information Processing Standards Publication 46. 1977.

[2] Schneier, B. “Applied Cryptography: Protocols, Algorithms, and Source Code in C”. John Wiley & Sons, 2007.

[3] Criptored. “Repositorio de Herramientas Criptográficas”. `http://www.criptored.upm.es/paginas/software.htm`, 2023.

[4] CrypTool Project. “CrypTool Documentation”. `https://www.cryptool.org/`, 2023.

[5] JCrypTool Team. “JCrypTool - Eclipse-based Crypto Toolkit”. `https://www.cryptool.org/en/jct/`, 2023.

[6] Singh, S. “The Code Book: The Science of Secrecy from Ancient Egypt to Quantum Cryptography”. Anchor Books, 2000.

[7] Lai, X., Massey, J. “A Proposal for a New Block Encryption Standard”. Advances in Cryptology — EUROCRYPT ’90. Lecture Notes in Computer Science, 1991.

[8] Katz, J., Lindell, Y. “Introduction to Modern Cryptography”. CRC Press, 2020.