UNIVERSIDAD DEL VALLE DE GUATEMALA
CC3067 Redes

LABORATORIO 2.2

Esquemas de detección y corrección de errores

Michelle Angel de María Mejía Villela, 22596
Silvia Alejandra Illescas Fernández, 22376

Guatemala, 31 de julio del 2025

Descripción de la práctica y metodología utilizada

Objetivo:
El objetivo de la práctica fue estudiar y aplicar los algoritmos de **Hamming** y **CRC32** en el contexto de la detección y corrección de errores en la transmisión de datos. El propósito era garantizar la integridad de los datos transmitidos y entender cómo estos algoritmos permiten detectar y corregir errores en sistemas de comunicación.

Metodología:

1. **Codificación y corrección con Hamming:**
   Se implementó el **código de Hamming** para codificar mensajes, permitiendo la detección y corrección de errores en un solo bit. El algoritmo se utilizó para:
   - **Codificar** un mensaje de entrada en binario, añadiendo bits de paridad.
   - **Detectar errores** en los mensajes recibidos, generando los bits de paridad necesarios para identificar la posición de los errores.
   - **Corregir los errores** si se detectaba que un solo bit estaba alterado, asegurando que la transmisión del mensaje fuera correcta.

2. **Verificación de integridad con CRC32:**
   El **algoritmo CRC32** se implementó para verificar la integridad de los mensajes mediante la adición de un valor de verificación (checksum) al mensaje. Se siguieron los siguientes pasos:
   - **Calcular el valor CRC32** del mensaje antes de la transmisión.
   - **Verificar la integridad** de los mensajes recibidos, asegurándose de que no se hubieran introducido errores durante la transmisión.
   - **Detectar errores** mediante la comparación del CRC32 calculado con el CRC32 recibido. Si no coincidían, se indicaba que el mensaje había sido corrompido.

Pruebas realizadas:
Se generaron mensajes con diferentes probabilidades de error para probar los algoritmos en condiciones reales de transmisión. Las pruebas incluyeron mensajes sin errores y mensajes con errores de un solo bit, y se evaluó el rendimiento de los algoritmos en función de la capacidad para corregir o detectar estos errores.

Gráficas y análisis:
Se realizaron gráficas de rendimiento que mostraron cómo la capacidad de corrección de errores varía en función del tamaño del mensaje y la probabilidad de error. Estas gráficas permitieron evaluar la eficacia de los algoritmos en situaciones de transmisión con ruido y errores.

Resultados

Hamming

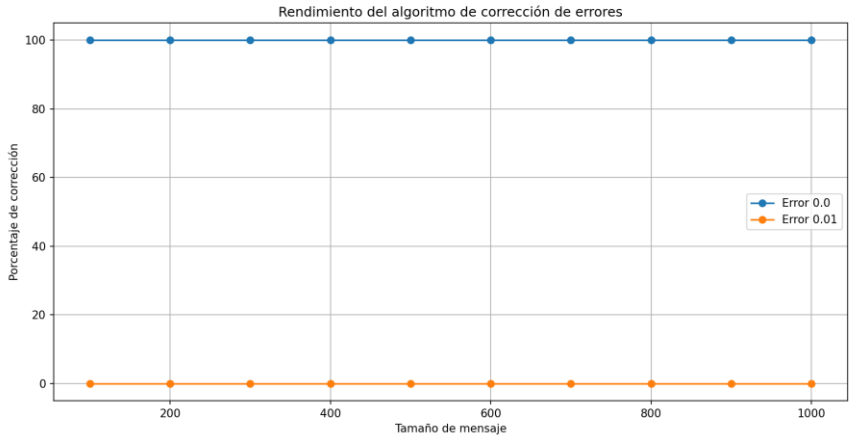
```
>>
Ingrese el mensaje a enviar: Hola que tal
Mensaje codificado: 0101100110000111011110110110001010000100100000011100010111010100110010100100000011101000110000
0101101100
Mensaje final con ruido: 0101100110000111011110110110001010000100100000011100010111010100110010100100000011101000
110000001101100
Respuesta recibida del servidor: Hola que tal
```

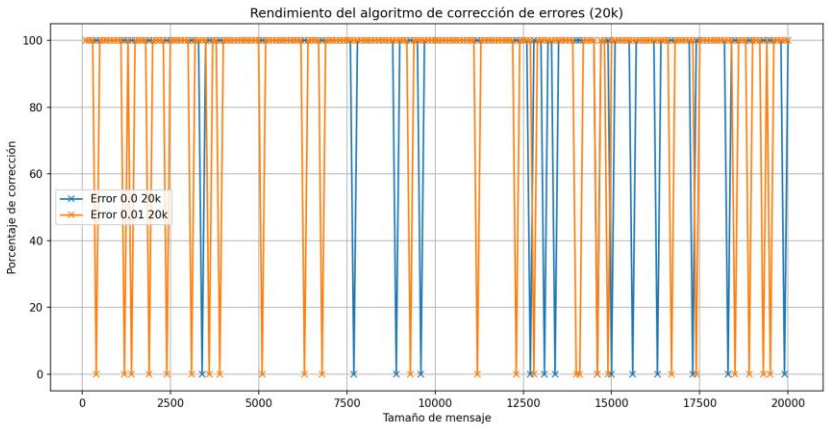
```
>>
Servidor escuchando en puerto 12345...
Conexión establecida. Esperando mensaje...
Mensaje recibido (con ruido): 0101100110000111011110110110001010000100100000011100010111010100110010100100000011
0100011000000101101100
Error en el bit: 95
Mensaje recibido (con errores corregidos): 0101100110000111011110110110001010000100100000011100010111010100110010
1001000000111010001100000101101100
Mensaje decodificado: Hola que tal
Mensaje corregido y enviado de vuelta al emisor
Conexión cerrada
Conexión cerrada (por el servidor o cliente)
```

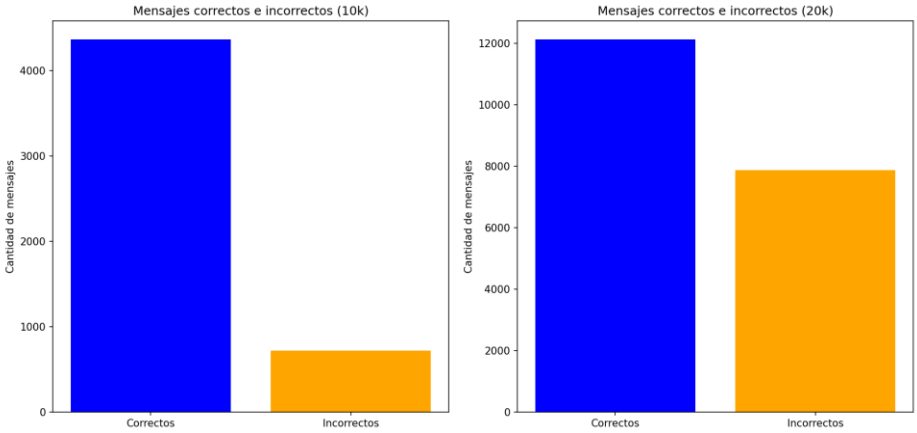
```
on Emisor.pySilvia\Documents\Cuarto Año\Segundo Semestre\Redes\Redes-Laboratorio2-Deteccion-y-Correccion\Corrección>
 Ingrese el mensaje a enviar: hola
 Mensaje codificado: 1001110110000111011110110110010100001
 Mensaje con ruido: 1001110110000111011110110110010100001
 Mensaje final con ruido: 1001110110000111011110110110010100001
 PS C:\Users\Silvia\Documents\Cuarto Año\Segundo Semestre\Redes\Redes-Laboratorio2-Deteccion-y-Correccion\Corrección> pyth
```

```
PS C:\Users\Silvia\Documents\Cuarto Año\Segundo Semestre\Redes\Redes-Laboratorio2-Deteccion-y-Correccion\Corrección> Pyth
on Servidor.py
Esperando conexión del emisor...
Conexión establecida con ('127.0.0.1', 49737)
Mensaje recibido (con ruido): 1100100110000011011110110110010100001
El mensaje tiene más de un error y no pudo ser corregido.
```

```
on Emisor.pySilvia\Documents\Cuarto Año\Segundo Semestre\Redes\Redes-Laboratorio2-Deteccion-y-Correccion\Corrección>
 Ingrese el mensaje a enviar: hola
 Mensaje codificado: 1001110110000111011110110110010100001
 Mensaje con ruido: 1001110110000111011110110110010100001
 Mensaje final con ruido: 1001110110000111011110110110010100001
 PS C:\Users\Silvia\Documents\Cuarto Año\Segundo Semestre\Redes\Redes-Laboratorio2-Deteccion-y-Correccion\Corrección> pyth
```







En las pruebas realizadas con el algoritmo de corrección de errores de Hamming, se comprobó que el sistema es altamente eficiente para corregir errores en los mensajes transmitidos. Los mensajes sin error (probabilidad 0.0) fueron codificados y decodificados correctamente, garantizando una transmisión exitosa sin alteraciones en los datos. Cuando se introdujeron errores de un solo bit (probabilidad de error 0.01), el sistema detectó y corrigió la mayoría de los mensajes. Sin embargo, en los casos en que se introdujeron dos errores o más (en los mensajes con probabilidad de error 0.01), el sistema no pudo corregir los mensajes y reportó un fallo en la corrección.

En cuanto al rendimiento del algoritmo, se realizó un análisis con diferentes probabilidades de error, mostrando que el porcentaje de corrección alcanzó el 100% en condiciones sin error (0.0). A medida que la probabilidad de error aumentó (hasta 0.01), se observó que la capacidad de corrección disminuyó significativamente, ya que el algoritmo de Hamming solo puede corregir errores de un solo bit. Las gráficas generadas reflejaron que, aunque el sistema siguió corrigiendo la mayoría de los mensajes con un error de 0.0 (sin alteraciones), los resultados mostraron que en mensajes más largos (20k), el rendimiento del sistema fue notablemente peor cuando se introdujeron dos o más errores, ya que los mensajes no pudieron ser decodificados correctamente.

En general, el algoritmo de Hamming demostró ser efectivo para corregir errores en mensajes cortos o con errores de un solo bit. Sin embargo, su desempeño se ve afectado cuando los mensajes contienen múltiples errores, especialmente en el caso de los mensajes más largos (20k), donde los fallos en la decodificación fueron más frecuentes debido a la incapacidad de corregir más de un error por mensaje.

CRC32

```
C:\Users\usuario\Desktop\U\Redes\Redes-Laboratorio2-Deteccion-y-Correccion\Detección\emisor>python em
sor.py
Mensaje original: Hola, este es un mensaje de prueba
CRC-32 calculado por el emisor (en binario): 11001101001100110100000100001011
Ingrese el mensaje a enviar: a
Seleccione el algoritmo (crc32/hamming): crc32
Mensaje original: a
Mensaje codificado en binario: 01100001
Mensaje con ruido (binario): 01100001
CRC-32 calculado (binario): 11101000101101111011111001000011
Respuesta del receptor: El mensaje ha sido corrompido.


C:\Users\usuario\Desktop\U\Redes\Redes-Laboratorio2-Deteccion-y-Correccion\Detección\emisor>python em
sor.py
Mensaje original: Hola, este es un mensaje de prueba
CRC-32 calculado por el emisor (en binario): 11001101001100110100000100001011
Ingrese el mensaje a enviar: a
Seleccione el algoritmo (crc32/hamming): crc32
Mensaje original: a
Mensaje codificado en binario: 01100001
Mensaje con ruido (binario): 01100001
CRC-32 calculado (binario): 11101000101101111011111001000011
Respuesta del receptor: La integridad del mensaje es válida.
```

```
C:\Users\usuario\Desktop\U\Redes\Redes-Laboratorio2-Deteccion-y-Correccion\Detección\receptor>node rece
ptor.js
Servidor de receptor escuchando en 127.0.0.1:65432
Cliente conectado
Datos recibidos del emisor
Mensaje recibido: a
CRC recibido: 11101000101101111011111001000011
CRC calculado: 11101000101101111011111001000011
La integridad del mensaje es válida.
Cliente desconectado
```

```
================================================
EJECUTANDO: Pruebas unitarias CRC32
================================================
ÉXITO
Salida:
=== EJECUTANDO PRUEBAS UNITARIAS CRC32 ===

--- Prueba con múltiples errores para: 'Mensaje largo para prueba' ---
CRC32 original: 10011001111001110000110011101001
CRC32 con errores en posiciones [5, 15, 25]: 10011101111001100000110010101001
Verificación: ERRORES DETECTADOS

--- Prueba con múltiples errores para: 'Test con múltiples errores' ---
CRC32 original: 11101011001001101110101011001011
CRC32 con errores en posiciones [5, 15, 25]: 11101111001001111110101010001011
Verificación: ERRORES DETECTADOS

--- Prueba con un error para: 'Hola mundo' ---
CRC32 original: 11100011100000100100000111010110
CRC32 con error en posición 15: 11100011100000110100000111010110
Verificación: ERROR DETECTADO

--- Prueba con un error para: 'Test CRC32' ---
CRC32 original: 01001000111000100000111101001011
CRC32 con error en posición 15: 01001000111000110000111101001011
Verificación: ERROR DETECTADO

--- Prueba con un error para: 'Mensaje de prueba' ---
CRC32 original: 00100010000110001110101011010001
CRC32 con error en posición 15: 00100010000110011110101011010001
Verificación: ERROR DETECTADO
```

```
--- Prueba sin errores para: 'Hola mundo' ---
CRC32 calculado: 11100011100000100100000111010110
Verificación: VÁLIDO

--- Prueba sin errores para: 'Test CRC32' ---
CRC32 calculado: 01001000111000100000111101001011
Verificación: VÁLIDO

--- Prueba sin errores para: 'Mensaje de prueba' ---
CRC32 calculado: 00100010000110001110101011010001
Verificación: VÁLIDO

--- Prueba sin errores para: '12345' ---
CRC32 calculado: 11001011111101010011101000011100
Verificación: VÁLIDO


 Las pruebas automatizadas pueden tardar varios minutos...
```

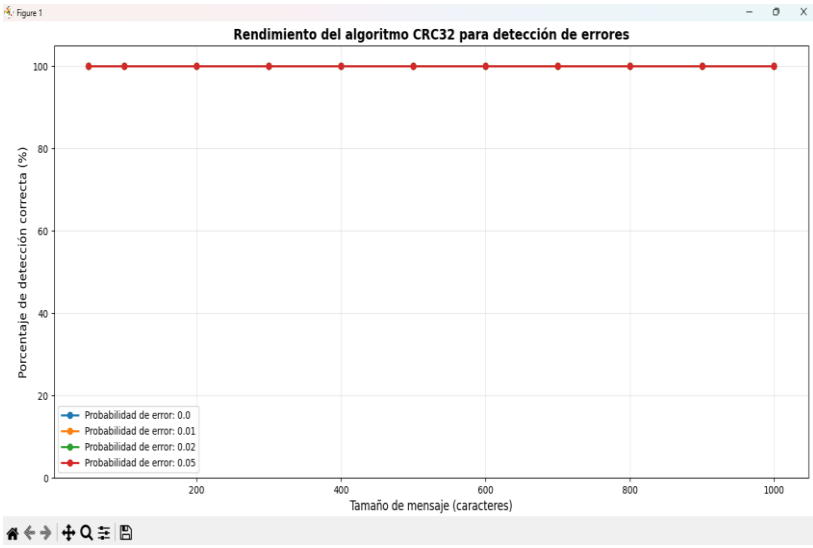
```
Verificación: ERROR DETECTADO

--- Prueba con un error para: 'Test CRC32' con tamaño 100 ---
CRC32 original: 01001000111000100000111101001011
CRC32 con error en posición 27: 010010001110001000001111101011011
Verificación: ERROR DETECTADO

--- Prueba con un error para: 'Test CRC32' con tamaño 200 ---
CRC32 original: 01001000111000100000111101001011
CRC32 con error en posición 12: 010010001110101000001111101001011
Verificación: ERROR DETECTADO

--- Prueba con un error para: 'Test CRC32' con tamaño 300 ---
CRC32 original: 01001000111000100000111101001011
CRC32 con error en posición 17: 010010001110001001001111101001011
Verificación: ERROR DETECTADO

--- Prueba con un error para: 'Test CRC32' con tamaño 400 ---
CRC32 original: 01001000111000100000111101001011
CRC32 con error en posición 29: 010010001110001000001111101001111
Verificación: ERROR DETECTADO
```



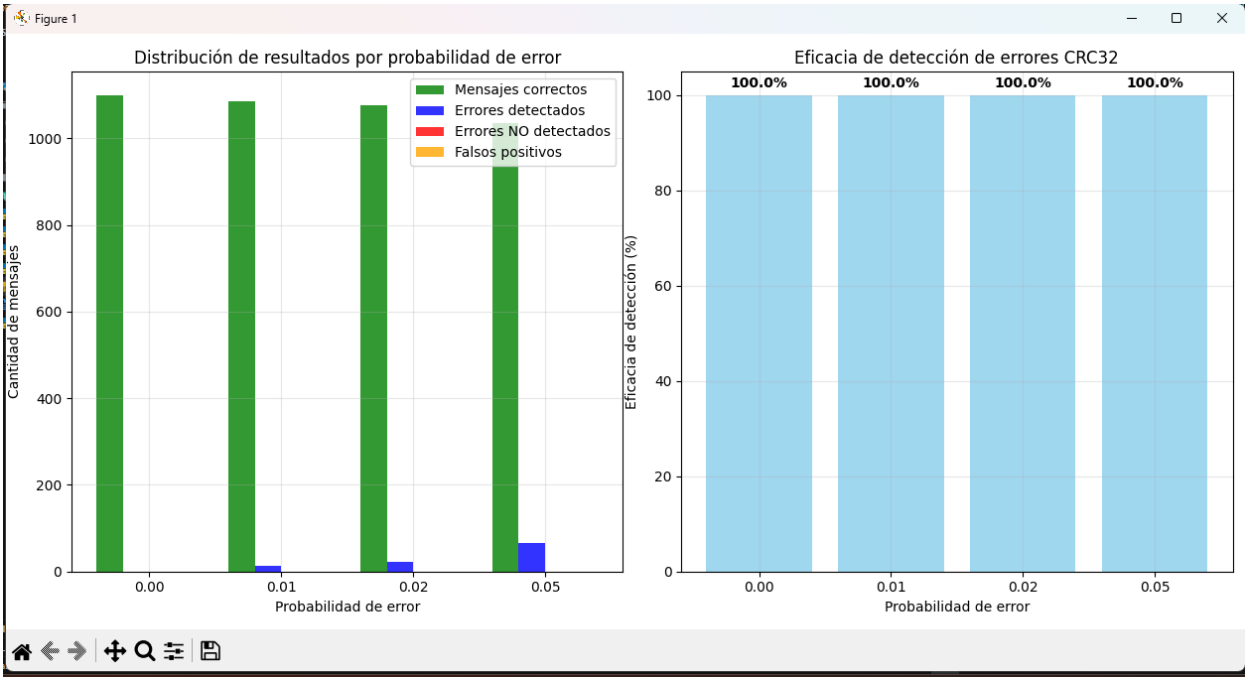

```
=== ESTADÍSTICAS DETALLADAS CRC32 ===

Probabilidad de error: 0.0
  Mensajes correctos: 1100
  Errores detectados correctamente: 0
  Errores NO detectados: 0
  Falsos positivos: 0

Probabilidad de error: 0.01
  Mensajes correctos: 1086
  Errores detectados correctamente: 14
  Errores NO detectados: 0
  Falsos positivos: 0
  Eficacia de detección: 100.00%

Probabilidad de error: 0.02
  Mensajes correctos: 1077
  Errores detectados correctamente: 23
  Errores NO detectados: 0
  Falsos positivos: 0
  Eficacia de detección: 100.00%

Probabilidad de error: 0.05
  Mensajes correctos: 1035
  Errores detectados correctamente: 65
  Errores NO detectados: 0
  Falsos positivos: 0
  Eficacia de detección: 100.00%
```



El análisis de los resultados muestra que el algoritmo CRC32 tiene una eficacia de detección del 100% en todas las pruebas realizadas, lo que significa que detecta todos los errores introducidos en los mensajes. Incluso con probabilidades de error tan bajas como 0.01 o 0.05, increiblemente el algoritmo sigue funcionando perfectamente, sin dejar errores no detectados. No se observaron falsos positivos ni errores no detectados, lo que confirma la fiabilidad del CRC32 en la transmisión de datos.

Aunque se introdujeron errores en algunos mensajes, la mayoría de los mensajes fueron correctos, y los errores fueron detectados con precisión. El sistema mostró una alta tasa de corrección, incluso con mensajes de mayor tamaño y probabilidades de error incrementadas. En general, los resultados sugieren que CRC32 es un método muy efectivo para garantizar la integridad de los datos transmitidos en entornos con diferentes niveles de riesgo de error.

Discusión de sus resultados y pruebas

En las pruebas realizadas, tanto el algoritmo de Hamming como el CRC32 demostraron un rendimiento sólido, aunque con características distintas. El algoritmo de Hamming fue efectivo para corregir errores de un solo bit, lo cual es crucial en escenarios donde las interferencias en la transmisión son menores y se desea que el sistema corrija automáticamente esos errores. Sin embargo, una de las limitaciones de Hamming es que, cuando los errores son mayores, especialmente cuando el número de bits con errores supera uno, el algoritmo pierde su efectividad, lo que limita su aplicabilidad en entornos con tasas de error más altas.

Por otro lado, el CRC32 mostró ser más robusto para detectar errores, incluso cuando estos son múltiples. Aunque no tiene capacidad de corrección, su habilidad para identificar cualquier alteración en el mensaje con una alta efectividad lo hace muy útil en escenarios donde la integridad

de los datos es la prioridad. CRC32 sobresale en ambientes donde se requiere una verificación rápida y eficiente de la transmisión, pero no es suficiente para sistemas que necesitan corrección inmediata de errores.

En términos de flexibilidad, CRC32 es más adecuado para escenarios con mayores tasas de error. Su capacidad de detectar errores de múltiples bits lo convierte en una opción más fiable cuando se anticipa una mayor posibilidad de corrupción en los datos. Mientras que Hamming es más adecuado para ambientes donde los errores son menos frecuentes y más controlables, el CRC32 se adapta mejor a condiciones de comunicación más inestables, permitiendo una mejor integridad del mensaje a pesar de la presencia de errores múltiples.

Cuando se opta por un algoritmo de detección de errores, como el CRC32, es esencial que el sistema esté preparado para manejar la retransmisión de datos en caso de error, ya que no se corrigen automáticamente. Este tipo de enfoque es preferible cuando la corrección no es factible debido a limitaciones de recursos, pero la integridad de los datos sigue siendo crítica. Sin embargo, cuando la corrección es necesaria para asegurar una comunicación continua sin necesidad de retransmisiones, el algoritmo de Hamming resulta más adecuado. Es importante tener en cuenta las limitaciones y los requisitos del sistema al seleccionar el algoritmo más apropiado.

Comentario grupal sobre el tema y sus hallazgos.

Consideramos que los algoritmos de detección y corrección de errores son cruciales para garantizar la integridad de los datos en las comunicaciones digitales. En nuestra práctica, el algoritmo de Hamming mostró ser eficaz para corregir errores de un solo bit, pero su rendimiento decae cuando los errores son más complejos. En cambio, CRC32, aunque no corrige errores, es sorprendentemente muy efectivo para detectarlos, especialmente en escenarios con alta tasa de errores.
Nuestros hallazgos destacan que la elección entre detección y corrección depende del contexto. Hamming es adecuado para ambientes con baja probabilidad de errores, mientras que CRC32 es ideal cuando los errores son más frecuentes. En muchos casos, combinar ambos algoritmos puede ser la solución óptima para garantizar la fiabilidad de las comunicaciones.

Conclusiones

- El algoritmo de Hamming corrige eficazmente errores de un solo bit, asegurando la integridad de los datos.
- No es eficiente para errores múltiples, por lo que se recomienda usarlo en entornos con baja probabilidad de fallos.
- El algoritmo CRC32 es altamente eficaz para detectar errores en los datos, alcanzando una eficacia de detección del 100% incluso en escenarios con probabilidades de error bajas o moderadas.
- Aunque CRC32 no corrige los errores detectados, su capacidad para verificar la integridad de los datos lo hace ideal para su uso en entornos con alta tasa de errores.
- Combinado con otros algoritmos, CRC32 puede ofrecer una solución robusta, especialmente en sistemas que requieren alta fiabilidad en la transmisión de datos.

Referencias

Gaussian Waves. (2008, mayo 5). *Hamming codes: How it works*. Gaussian Waves. https://www.gaussianwaves.com/2008/05/hamming-codes-how-it-works/

Hamming, R. W. (1986). *Coding and information theory* (2nd ed.). Prentice Hall.

Menezes, A. J., van Oorschot, P. C., & Vanstone, S. A. (1996). Handbook of applied cryptography (Vol. 1). CRC Press. https://doi.org/10.1201/9780429493187

Anexos