UNIVERSIDAD CENTRAL DEL ECUADOR

FACULTAD DE INGENIERÍA Y CIENCIAS APLICADAS

CRIPTOGRAFÍA Y SEGURIDAD

GRUPO #3

ALGORITMOS CRIPTOGRÁFICOS REALIZADOS EN PYTHON

INTEGRANTES:

- **ANDRADE DENNISSE**
- **LUGMAÑA ÁNDRES**
- **OCAPANA JOSUE**
- **NARVÁEZ ANTHONY**
- **RIVAS DIEGO**

2024

INTRODUCCIÓN

1. Algoritmos Simétricos:

Son métodos criptográficos donde se utiliza la misma clave tanto para cifrar como para descifrar la información. Es decir, la misma clave se usa para ambas direcciones del proceso de encriptación. Ejemplos comunes son AES (Advanced Encryption Standard) y DES (Data Encryption Standard).

2. Algoritmos Asimétricos:

También conocidos como criptografía de clave pública, son métodos donde se utilizan dos claves diferentes pero relacionadas: una clave pública y una clave privada. La clave pública se comparte abiertamente, mientras que la clave privada se mantiene secreta. Ejemplos incluyen RSA (Rivest-Shamir-Adleman) y ECC (ElliMptic Curve Cryptography).

3. Funciones Hash:

Son algoritmos que toman una cantidad variable de datos y generan una cadena de caracteres de longitud fija. Estas funciones son unidireccionales, lo que significa que no se puede obtener la entrada original a partir de la salida (el hash). Se utilizan en la verificación de integridad de datos y la creación de firmas digitales. Ejemplos populares son SHA-256 y MD5.

DESAROLLO DE LOS ALGORITMOS:

1. **RC4**

   Es un algoritmo de flujo de cifrado simétrico, diseñado por Ron Rivest en 1987. Utiliza una clave variable de longitud y opera mediante una mezcla de permutaciones y sustituciones. RC4 es rápido y simple de implementar, lo que lo ha hecho ampliamente utilizado en aplicaciones que requieren cifrado de datos en tiempo real, como SSL/TLS y WEP/WPA en redes Wi-Fi. Sin embargo, se han descubierto vulnerabilidades en su seguridad, y actualmente se recomienda evitar su uso en nuevos sistemas a favor de algoritmos más seguros.

   Ejecución:

- **10**

```
Tiempo de lectura del archivo: 1.00 milisegundos
Tiempo de generación de la clave: 0.00 milisegundos
Clave de cifrado: 95379131140a526013e1c704d75b815d
Longitud del texto original: 60
Tiempo de cifrado: 0.00 milisegundos
Longitud del texto cifrado: 60
Tiempo de descifrado: 0.00 milisegundos
```

- 100

```
Tiempo de lectura del archivo: 12.56 milisegundos
Tiempo de generación de la clave: 0.00 milisegundos
Clave de cifrado: 9fbb2fc52b4f65a7d5af638508a280c9
Longitud del texto original: 599
Tiempo de cifrado: 0.00 milisegundos
Longitud del texto cifrado: 599
Tiempo de descifrado: 0.00 milisegundos
```

- 1000

```
Tiempo de lectura del archivo: 31.18 milisegundos
Tiempo de generación de la clave: 0.00 milisegundos
Clave de cifrado: 1c1ab9490f5fdbb2db10a99233747541
Longitud del texto original: 6337
Tiempo de cifrado: 0.00 milisegundos
Longitud del texto cifrado: 6337
Tiempo de descifrado: 0.00 milisegundos
```

- 10000

```
Tiempo de lectura del archivo: 34.70 milisegundos
Tiempo de generación de la clave: 0.00 milisegundos
Clave de cifrado: 0fe570145082be1a69c63ded545d1b41
Longitud del texto original: 57943
Tiempo de cifrado: 1.01 milisegundos
Longitud del texto cifrado: 57943
Tiempo de descifrado: 0.00 milisegundos
```

- 100000

```
Tiempo de lectura del archivo: 44.20 milisegundos
Tiempo de generación de la clave: 0.00 milisegundos
Clave de cifrado: 7fbf0b5ea3539cbf7d1aa7bd10504707
Longitud del texto original: 580629
Tiempo de cifrado: 5.00 milisegundos
Longitud del texto cifrado: 580629
Tiempo de descifrado: 4.99 milisegundos
```

- 1000000

```
Tiempo de lectura del archivo: 205.78 milisegundos
Tiempo de generación de la clave: 0.00 milisegundos
Clave de cifrado: 0f8f957e6b8c525d829479b2164b2f89
Longitud del texto original: 5806306
Tiempo de cifrado: 21.99 milisegundos
Longitud del texto cifrado: 5806306
Tiempo de descifrado: 21.99 milisegundos
```

- 10000000

```
Tiempo de lectura del archivo: 1284.40 milisegundos
Tiempo de generación de la clave: 0.00 milisegundos
Clave de cifrado: 706c6bb27deb1c44edbfb9fb42c5f4ed
Longitud del texto original: 58063060
Tiempo de cifrado: 215.87 milisegundos
Longitud del texto cifrado: 58063060
Tiempo de descifrado: 218.87 milisegundos
```

Ventaja:
RC4 es conocido por su rapidez de cifrado, lo que lo hace ideal para aplicaciones que requieren un procesamiento eficiente de grandes volúmenes de datos.

Desventaja:
Sin embargo, RC4 ha sufrido múltiples ataques y se considera inseguro para aplicaciones modernas por vulnerabilidades conocidas que pueden comprometer la seguridad de los datos cifrados.

2. **Blowfish**
   Es un algoritmo de cifrado simétrico de bloque diseñado por Bruce Schneier en 1993 como un reemplazo rápido y seguro para DES (Data Encryption Standard). Utiliza bloques de datos de 64 bits y una clave variable de longitud, que puede ser de 32 a 448 bits. Blowfish opera mediante una serie de rondas de cifrado que involucran permutaciones y sustituciones. Aunque ha sido ampliamente estudiado y no se han encontrado vulnerabilidades significativas, su uso generalmente se ha visto eclipsado por AES (Advanced Encryption Standard) en aplicaciones modernas debido a las recomendaciones de cifrado.

Ejecución:

- **10**

```
PS D:\josue\Escritorio\Josue\Computacion R\8vo\Cripto\Tarea08-U1-G3> & C:/Users/josue/AppData/Local/Microsoft/WindowsApps/PythonSoftwareFoundation.Python.3.12_qbz5n2kfra8p0/python.exe "d:/josue/Escritorio/Josue/Computacion R/8vo/Cripto/Tarea08-U1-G3/algoritmoblowfish.py"

Tiempo para leer el archivo: 0.0 milisegundos

Clave generada: b'J\xd02\x9fx\xb1\xe5m'
Tiempo para generar la clave: 0.0 milisegundos

Texto cifrado: b'\xb0\xc7\x8d\x99r-\xd5\xed\xc9B\x9a\x81]\x84E\x0cWf\x9cnB\x19l;l\xa0.[[\x8d\x04\xc8\n\x95/s\xa2\x04z=\xaeft\xaaY\x82\xd9:\xf4\xa9}ov\xea\xdd\xfb\xc8\x08L\xe3\xd9|\xa3\x14'
Tiempo para cifrar el texto: 0.0 milisegundos

Texto descifrado: Claridad en el horizonte, brilla el sol, renace la esperanza.
Tiempo para descifrar el texto: 0.0 milisegundos

Caracteres en el archivo: 52

Caracteres en el archivo cifrado: 64

PS D:\josue\Escritorio\Josue\Computacion R\8vo\Cripto\Tarea08-U1-G3>
```

- **100**

```
on.exe "d:/josue/Escritorio/Josue/Computacion R/8vo/Cripto

Tiempo para leer el archivo: 0.0 milisegundos

Clave generada: b'\xecm&\x03QRDR'
Tiempo para generar la clave: 0.0 milisegundos
Tiempo para cifrar el texto: 0.0 milisegundos
Tiempo para descifrar el texto: 0.0 milisegundos

Caracteres en el archivo: 562

Caracteres en el archivo cifrado: 672
```

- **1000**

```
on.exe "d:/josue/Escritorio/Josue/Computacion R/8vo/Cripto/Tarea

Tiempo para leer el archivo: 1.0745525360107422 milisegundos

Clave generada: b'*(D\xd8\x07\x9a\xae\xa6'
Tiempo para generar la clave: 0.9238719940185547 milisegundos
Tiempo para cifrar el texto: 0.0 milisegundos
Tiempo para descifrar el texto: 0.0 milisegundos

Caracteres en el archivo: 5500

Caracteres en el archivo cifrado: 6640
```

- **10000**

```
Tiempo para leer el archivo: 23.47278594970703 milisegundos

Clave generada: b'\xef\xaa\x8e\x0f4\xc3>\x13'
Tiempo para generar la clave: 0.0 milisegundos
Tiempo para cifrar el texto: 0.99945068359375 milisegundos
Tiempo para descifrar el texto: 1.0094642639160156 milisegundos

Caracteres en el archivo: 63328

Caracteres en el archivo cifrado: 73328

PS D:\josue\Escritorio\Josue\Computacion R\8vo\Cripto\Tarea08-U1-G3>
```

- **100000**

```
Tiempo para leer el archivo: 36.05461120605469 milisegundos

Clave generada: b'H\xf0\xaej\x9dT\xbd\xfb'
Tiempo para generar la clave: 0.0 milisegundos
Tiempo para cifrar el texto: 5.651235580444336 milisegundos
Tiempo para descifrar el texto: 6.166696548461914 milisegundos

Caracteres en el archivo: 633280

Caracteres en el archivo cifrado: 733264
```

- **1000000**

```
Tiempo para leer el archivo: 109.24553871154785 milisegundos

Clave generada: b'\x98o_\xe0\xd7\xf5>\xd8'
Tiempo para generar la clave: 0.0 milisegundos
Tiempo para cifrar el texto: 57.56211280822754 milisegundos
Tiempo para descifrar el texto: 50.69851875305176 milisegundos

Caracteres en el archivo: 6332800

Caracteres en el archivo cifrado: 7332608
```

- **10000000**

```
Tiempo para leer el archivo: 949.97239112854 milisegundos

Clave generada: b'\xbf\x93\\A.ZB\xdd'
Tiempo para generar la clave: 0.0 milisegundos
Tiempo para cifrar el texto: 580.1093578338623 milisegundos
Tiempo para descifrar el texto: 601.8779277801514 milisegundos

Caracteres en el archivo: 63328000

Caracteres en el archivo cifrado: 73326008
```

Ventaja:

Blowfish es un algoritmo de cifrado sólido que ha resistido el tiempo y se considera seguro para su uso en una amplia gama de aplicaciones. Además, es eficiente en términos de velocidad y consumo de recursos.

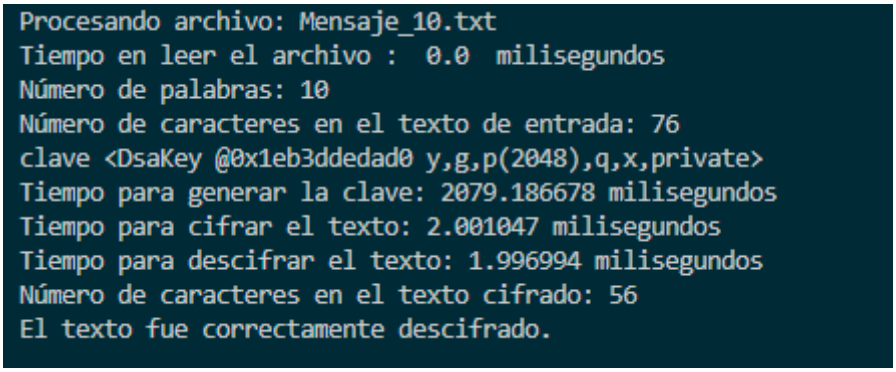
Desventaja:

Aunque Blowfish es robusto, su velocidad puede ser superada por algoritmos más modernos.

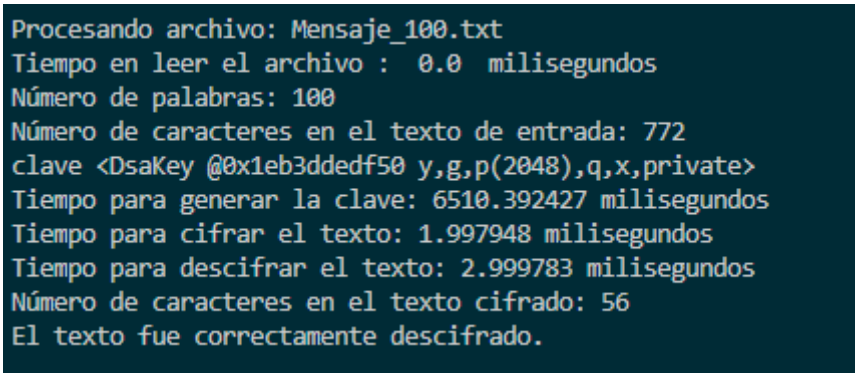
3. **DSA (Digital Signature Algorithm):**
   Es un algoritmo de firma digital utilizado para garantizar la autenticidad, integridad y no repudio de mensajes digitales. Fue propuesto por el Instituto Nacional de Normas y Tecnología (NIST) de los Estados Unidos en 1991 como un estándar de firma digital. DSA utiliza operaciones matemáticas basadas en el problema del logaritmo discreto sobre un grupo multiplicativo finito, típicamente en el contexto del grupo de números enteros módulo un primo grande. Se usa en protocolos criptográficos como DSA y SSH para dar autenticación y firmas digitales. Sin embargo, se debe tener cuidado en su implementación para evitar vulnerabilidades relacionadas con el tamaño de los parámetros y el proceso de generación de claves.

   Ejecución:

   - **10**



   - **100**



   - **1000**

```
Procesando archivo: Mensaje_1000.txt
Tiempo en leer el archivo :  1.001596450805664  milisegundos
Número de palabras: 1000
Número de caracteres en el texto de entrada: 7527
clave <DsaKey @0x1eb3ddec710 y,g,p(2048),q,x,private>
Tiempo para generar la clave: 20198.308229 milisegundos
Tiempo para cifrar el texto: 1.994133 milisegundos
Tiempo para descifrar el texto: 4.138947 milisegundos
Número de caracteres en el texto cifrado: 56
El texto fue correctamente descifrado.
```

- **10000**

```
Procesando archivo: Mensaje_10000.txt
Tiempo en leer el archivo :  1.003265380859375  milisegundos
Número de palabras: 10000
Número de caracteres en el texto de entrada: 75353
clave <DsaKey @0x1eb3ddede90 y,g,p(2048),q,x,private>
Tiempo para generar la clave: 18676.829100 milisegundos
Tiempo para cifrar el texto: 2.000809 milisegundos
Tiempo para descifrar el texto: 3.006220 milisegundos
Número de caracteres en el texto cifrado: 56
El texto fue correctamente descifrado.
```

- **100000**

```
Procesando archivo: Mensaje_100000.txt
Tiempo en leer el archivo :  4.997014999389648  milisegundos
Número de palabras: 100000
Número de caracteres en el texto de entrada: 749822
clave <DsaKey @0x1eb3ddee150 y,g,p(2048),q,x,private>
Tiempo para generar la clave: 530.352831 milisegundos
Tiempo para cifrar el texto: 3.996134 milisegundos
Tiempo para descifrar el texto: 7.000923 milisegundos
Número de caracteres en el texto cifrado: 56
El texto fue correctamente descifrado.
```

- **1000000**

```
Procesando archivo: Mensaje_1000000.txt
Tiempo en leer el archivo :  52.00648307800293  milisegundos
Número de palabras: 1000000
Número de caracteres en el texto de entrada: 7497648
clave <DsaKey @0x1eb3ddec910 y,g,p(2048),q,x,private>
Tiempo para generar la clave: 1012.366056 milisegundos
Tiempo para cifrar el texto: 29.997587 milisegundos
Tiempo para descifrar el texto: 32.007217 milisegundos
Número de caracteres en el texto cifrado: 56
El texto fue correctamente descifrado.
```

- **10000000**

```
Procesando archivo: Mensaje_10000000.txt
Tiempo en leer el archivo :  525.5303382873535  milisegundos
Número de palabras: 10000000
Número de caracteres en el texto de entrada: 75005603
clave <DsaKey @0x1eb3ddec110 y,g,p(2048),q,x,private>
Tiempo para generar la clave: 1634.775162 milisegundos
Tiempo para cifrar el texto: 285.898447 milisegundos
Tiempo para descifrar el texto: 299.234629 milisegundos
Número de caracteres en el texto cifrado: 56
El texto fue correctamente descifrado.
PS C:\Users\PC\Documents\pythonCripto>
```

Ventaja:
DSA se usa mucho en la firma digital y proporciona un buen equilibrio entre seguridad y eficiencia computacional.

Desventaja:
DSA tiene limitaciones en cuanto al tamaño de las claves y los parámetros, lo que puede afectar su seguridad en entornos donde se requiere una mayor resistencia a los ataques.

4. **ElGamal**
   Es un algoritmo de cifrado de clave pública y firma digital, nombrado en honor al criptógrafo Taher Elgamal, quien lo describió por primera vez en 1985. ElGamal se basa en la dificultad de resolver el problema del logaritmo discreto en grupos cíclicos finitos. Este algoritmo se usa para cifrado de mensajes y para firmar digitalmente mensajes, proporcionando confidencialidad e integridad en comunicaciones seguras.

   En el cifrado de mensajes, ElGamal opera sobre grupos cíclicos finitos, como grupos de números enteros módulo un primo grande. Se basa en la dificultad de calcular logaritmos discretos en estos grupos. La seguridad del cifrado depende de la dificultad de este problema.

   ElGamal también se utiliza en protocolos de firma digital, donde se generan claves públicas y privadas. La clave pública se utiliza para verificar la autenticidad de una firma digital, mientras que la clave privada se utiliza para crear firmas digitales.

   Ejecución:
   - **10**

```
Procesando archivo: Mensaje_10.txt
Tiempo en leer el archivo :  0.0  milisegundos
Número de palabras: 10
Número de caracteres en el texto de entrada: 76
Tiempo transcurrido en generación de claves: 0.0 milisegundoss
Número de caracteres en el texto cifrado: 76
Tiempo transcurrido en cifrado: 0.0 milisegundos
Número de caracteres en el texto descifrado: 76
Tiempo transcurrido en descifrado: 0.0 milisegundos
```

- **100**

```
Procesando archivo: Mensaje_100.txt
Tiempo en leer el archivo :  0.9973049163818359  milisegundos
Número de palabras: 100
Número de caracteres en el texto de entrada: 772
Tiempo transcurrido en generación de claves: 0.0 milisegundoss
Número de caracteres en el texto cifrado: 772
Tiempo transcurrido en cifrado: 0.0 milisegundos
Número de caracteres en el texto descifrado: 772
Tiempo transcurrido en descifrado: 1.9936561584472656 milisegundos
```

- **1000**

```
Procesando archivo: Mensaje_1000.txt
Tiempo en leer el archivo :  0.0  milisegundos
Número de palabras: 1000
Número de caracteres en el texto de entrada: 7527
Tiempo transcurrido en generación de claves: 0.0 milisegundoss
Número de caracteres en el texto cifrado: 7527
Tiempo transcurrido en cifrado: 5.005121231079102 milisegundos
Número de caracteres en el texto descifrado: 7527
Tiempo transcurrido en descifrado: 29.000520706176758 milisegundos
```

- **10000**

```
Procesando archivo: Mensaje_10000.txt
Tiempo en leer el archivo :  1.2552738189697266  milisegundos
Número de palabras: 10000
Número de caracteres en el texto de entrada: 75353
Tiempo transcurrido en generación de claves: 0.0 milisegundoss
Número de caracteres en el texto cifrado: 75353
Tiempo transcurrido en cifrado: 46.26941680908203 milisegundos
Número de caracteres en el texto descifrado: 75353
Tiempo transcurrido en descifrado: 211.10224723815918 milisegundos
```

- **100000**

```
Procesando archivo: Mensaje_100000.txt
Tiempo en leer el archivo :  4.995107650756836  milisegundos
Número de palabras: 100000
Número de caracteres en el texto de entrada: 749822
Tiempo transcurrido en generación de claves: 0.0 milisegundoss
Número de caracteres en el texto cifrado: 749822
Tiempo transcurrido en cifrado: 418.7021255493164 milisegundos
Número de caracteres en el texto descifrado: 749822
Tiempo transcurrido en descifrado: 1989.5732402801514 milisegundos
```

- **1000000**

```
Procesando archivo: Mensaje_1000000.txt
Tiempo en leer el archivo :  50.47726631164551  milisegundos
Número de palabras: 1000000
Número de caracteres en el texto de entrada: 7497648
Tiempo transcurrido en generación de claves: 0.0 milisegundoss
Número de caracteres en el texto cifrado: 7497648
Tiempo transcurrido en cifrado: 2540.297031402588 milisegundos
Número de caracteres en el texto descifrado: 7497648
Tiempo transcurrido en descifrado: 20746.333837509155 milisegundos
```

- **10000000**

```
Procesando archivo: Mensaje_10000000.txt
Tiempo en leer el archivo :  494.33112144470215  milisegundos
Número de palabras: 10000000
Número de caracteres en el texto de entrada: 75005603
Tiempo transcurrido en generación de claves: 0.0 milisegundoss
Número de caracteres en el texto cifrado: 75005603
Tiempo transcurrido en cifrado: 40327.66890525818 milisegundos
Número de caracteres en el texto descifrado: 75005603
Tiempo transcurrido en descifrado: 456013.67354393005 milisegundos
PS C:\Users\PC\Documents\pythonCripto>
```

Ventaja:

ElGamal es uno de los pocos algoritmos de cifrado de clave pública que ofrece tanto cifrado como firma digital en un solo protocolo, lo que lo hace versátil en una variedad de aplicaciones criptográficas.

Desventaja:

Sin embargo, ElGamal puede ser más lento que otros algoritmos, especialmente en comparación con sistemas basados en curvas elípticas como ECC, lo que puede afectar su rendimiento en ciertos escenarios.

5. **Whirlpool**

   Es una familia de funciones hash criptográficas diseñada por Vincent Rijmen y Paulo S. L. M. Barreto. Fue seleccionada como una de las funciones hash estándar en el concurso de Funciones Hash Criptográficas

del Instituto Nacional de Estándares y Tecnología (NIST) de los Estados Unidos. Whirlpool es para resistir a los ataques criptográficos, incluidos ataques de colisión y preimagen.

Esta función hash produce un resumen criptográfico de un conjunto de datos de entrada de longitud variable. Utiliza una estructura de red de Feistel modificada, operaciones no lineales y una serie de rondas para mezclar y transformar los datos de entrada. El resultado es un valor hash de longitud fija que es único para un conjunto de datos dado, lo que lo hace útil para verificar la integridad de datos y la autenticación de mensajes.

Whirlpool tiene diferentes variantes que producen valores hash de diferentes longitudes, como Whirlpool-0 (128 bits), Whirlpool-T (192 bits) y Whirlpool (512 bits). La versión más utilizada es Whirlpool, que produce un valor hash de 512 bits.

Ejecución:

- **10**

```
[Running] python -u "d:\OCTAVO\CRIPTO\Whirlpool\funcionWhirlpool.py"
Tiempo de lectura del archivo: 0.0 milisegundos
Cantidad de caracteres le□dos: 56
Tiempo de generaci□n del cifrado: 8.610963821411133 milisegundos
Cantidad de caracteres del cifrado: 128
Clave de cifrado (Whirlpool): efaccb8a8d03bf00adec21c1663104eaee02b18b8099f96c5ac3
```

- **100**

```
[Running] python -u "d:\OCTAVO\CRIPTO\Whirlpool\funcionWhirlpool.py"
Tiempo de lectura del archivo: 0.0 milisegundos
Cantidad de caracteres le□dos: 776
Tiempo de generaci□n del cifrado: 0.9229183197021484 milisegundos
Cantidad de caracteres del cifrado: 128
Clave de cifrado (Whirlpool): 9e31e32bcbacf6b9b60b02129b3d9dc59f870b8b57217b24066e3fb0f556
```

- **1000**

```
[Running] python -u "d:\OCTAVO\CRIPTO\Whirlpool\funcionWhirlpool.py"
Tiempo de lectura del archivo: 0.0 milisegundos
Cantidad de caracteres le□dos: 7339
Tiempo de generaci□n del cifrado: 0.0 milisegundos
Cantidad de caracteres del cifrado: 128
Clave de cifrado (Whirlpool): 2507660d1dc10f6bf4e03df9018571a918ba3420e1df44
```

- **10000**

```
[Running] python -u "d:\OCTAVO\CRIPTO\Whirlpool\funcionWhirlpool.py"
Tiempo de lectura del archivo: 0.0 milisegundos
Cantidad de caracteres le�dos: 74881
Tiempo de generaci�n del cifrado: 0.0 milisegundos
Cantidad de caracteres del cifrado: 128
Clave de cifrado (Whirlpool): 099667b45c1f0dc67e0fda17ab9700ec71abc8ce666
```

- **100000**

```
[Running] python -u "d:\OCTAVO\CRIPTO\Whirlpool\funcionWhirlpool.py"
Tiempo de lectura del archivo: 0.0 milisegundos
Cantidad de caracteres le�dos: 751706
Tiempo de generaci�n del cifrado: 15.651464462280273 milisegundos
Cantidad de caracteres del cifrado: 128
Clave de cifrado (Whirlpool): 29996f44df452ca5020c9ba67598c3d774d9c425055f89
```

- **1000000**

```
[Running] python -u "d:\OCTAVO\CRIPTO\Whirlpool\funcionWhirlpool.py"
Tiempo de lectura del archivo: 59.17692184448242 milisegundos
Cantidad de caracteres le�dos: 7500982
Tiempo de generaci�n del cifrado: 72.9517936706543 milisegundos
Cantidad de caracteres del cifrado: 128
Clave de cifrado (Whirlpool): 4fe0cb6375d891a9ef4c21c62759891255da82d788ca3ceb4ec5
```

- **10000000**

```
[Running] python -u "d:\OCTAVO\CRIPTO\Whirlpool\funcionWhirlpool.py"
Tiempo de lectura del archivo: 483.72793197631836 milisegundos
Cantidad de caracteres le�dos: 74986205
Tiempo de generaci�n del cifrado: 501.3413429260254 milisegundos
Cantidad de caracteres del cifrado: 128
Clave de cifrado (Whirlpool): 4b5f65cdd473171aee8039e35a650bf716241920e841
```

Ventaja:
Whirlpool es una función hash robusta que ofrece una alta seguridad contra colisiones y ataques criptográficos conocidos.

Desventaja:
Aunque Whirlpool es seguro, puede ser menos eficiente en términos de velocidad y consumo de recursos en comparación con funciones hash más simples como SHA-256.

6. **RIPEMD (RACE Integrity Primitives Evaluation Message Digest)**
   Es una familia de funciones hash criptográficas desarrollada por Hans Dobbertin, Antoon Bosselaers y Bart Preneel en la década de 1990 como

una alternativa al algoritmo MD4. Fue diseñada inicialmente en el marco del proyecto europeo RACE para garantizar la integridad de mensajes en sistemas de comunicación.

Las variantes de RIPEMD, como RIPEMD-128, RIPEMD-160, RIPEMD-256 y RIPEMD-320, producen resúmenes criptográficos de diferentes longitudes, desde 128 bits hasta 320 bits. Estas funciones hash se basan en principios similares a otras funciones hash, como MD4 y SHA-1, pero utilizan operaciones más sofisticadas y rondas adicionales para mejorar la seguridad.

RIPEMD se utiliza en una variedad de aplicaciones, incluyendo la verificación de integridad de datos, la autenticación de mensajes y la generación de claves criptográficas. Aunque RIPEMD ha resistido varios ataques criptográficos y se considera seguro en muchos contextos, su uso en nuevas aplicaciones puede verse afectado por avances en la criptografía y la preferencia por funciones hash más modernas, como SHA-256 y SHA-3.

RIMPED 160:

Ejecución:

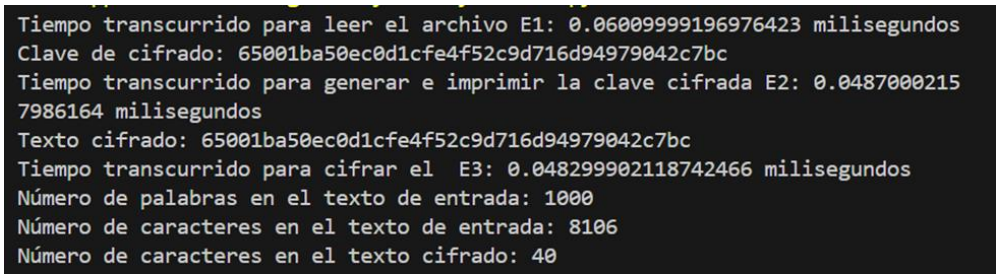
- **10**

```
Tiempo transcurrido para leer el archivo E1: 0.021000043489038944 milisegundos
Clave de cifrado: 17d643967edbd05ec08ee76f744bc8a3c2e60720
Tiempo transcurrido para generar e imprimir la clave cifrada E2: 0.010999967344105244 milisegundos
Texto cifrado: 17d643967edbd05ec08ee76f744bc8a3c2e60720
Tiempo transcurrido para cifrar el  E3: 0.01279998105019331 milisegundos
Número de palabras en el texto de entrada: 10
Número de caracteres en el texto de entrada: 77
Número de caracteres en el texto cifrado: 40
```

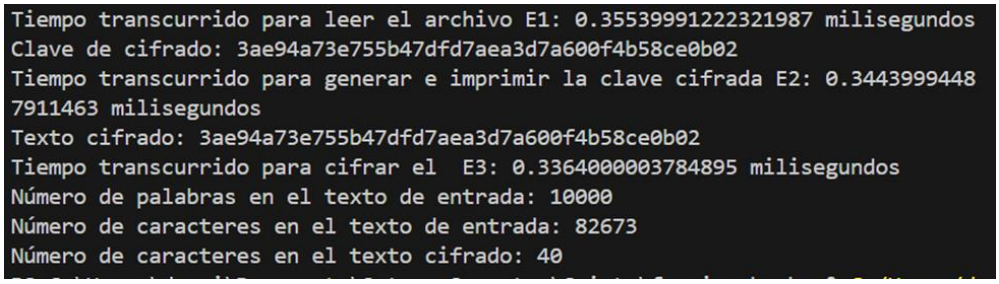
- **100**

```
Tiempo transcurrido para leer el archivo E1: 0.03769993782043457 milisegundos
Clave de cifrado: 841ab1be4363fc94c96a03acf2529733812152d3
Tiempo transcurrido para generar e imprimir la clave cifrada E2: 0.0238999491
5574789 milisegundos
Texto cifrado: 841ab1be4363fc94c96a03acf2529733812152d3
Tiempo transcurrido para cifrar el  E3: 0.026899971999228 milisegundos
Número de palabras en el texto de entrada: 100
Número de caracteres en el texto de entrada: 762
Número de caracteres en el texto cifrado: 40
```

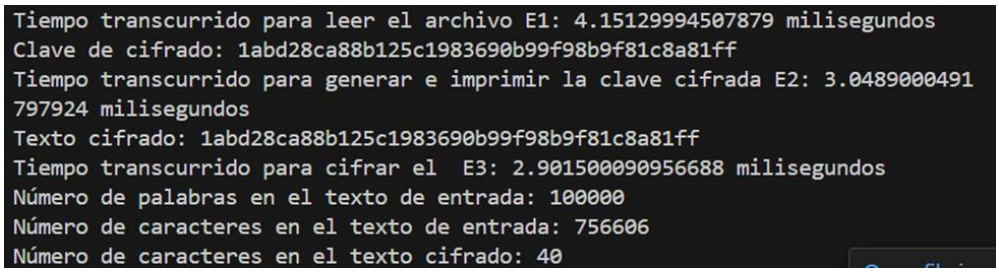
- **1000**



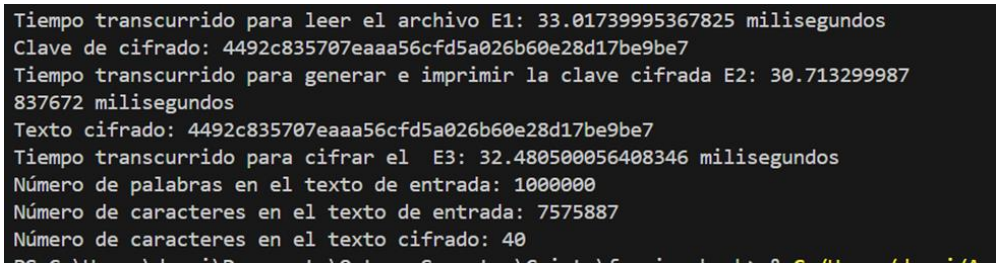
- **10000**



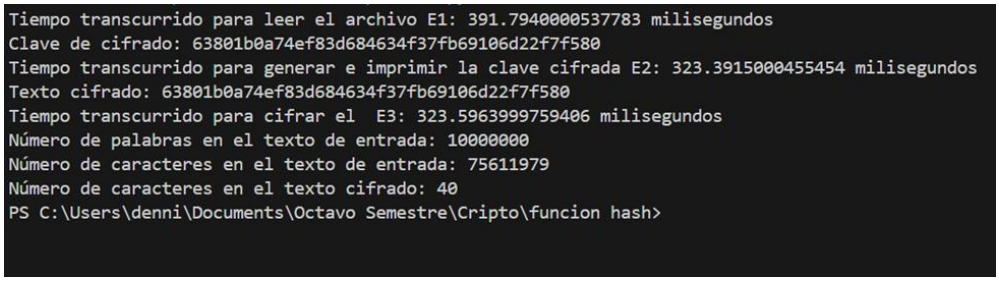
- **100000**



- **1000000**



- **10000000**



Ventaja:

RIPEMD es una función hash rápida y eficiente que ofrece una buena seguridad para la mayoría de las aplicaciones criptográficas.

Desventaja:
Sin embargo, RIPEMD puede ser menos resistente a ciertos ataques en comparación con funciones hash más modernas como SHA-3, especialmente en entornos donde se requiere una alta seguridad.

ANALISIS:
RC4 es conocido por su rápida implementación por su diseño sencillo, pero su seguridad comprometida lo hace poco recomendable en los resultados que nos votó se puedo verificar la efectividad de este para cifrar, ya que los recursos que maneja son pocos.

De hecho, su implementación fue bastante sencilla de ejecutar Por otro lado, Blowfish ofrece un buen equilibrio entre seguridad y eficiencia, el código fue un poco más complejo de llevar a cabo, pero los resultados fueron en cierta parte bastante similares al anterior, pero con unas ligeras diferencias lo que lo hace versátil es su capacidad mínimamente más eficiente que RC4. Aunque en cuanto a su seguridad cuenta con un mejor algoritmo al comparar sus claves con el RC4 está parece ser más robusta, su velocidad puede ser superada por algoritmos más adelante se realizaron su respectiva ejecución.

DSA proporciona una seguridad razonable con eficiencia computacional, ya que los resultados fueron diferentes en comparación a los algoritmos simétricos, siendo diferente incluso en el manejo de las claves para realizar el cifrado y el descifrado.

No obstante, sus limitaciones en el tamaño de clave y parámetros pueden afectar su aplicabilidad lo cual se puede evidenciar entre las diferentes ejecuciones del programa, ElGamal, aunque más complejo de implementar debido a su naturaleza de clave pública siendo uno de los algoritmos que más complicado fue en realizar, se dice que este algoritmo ofrece tanto cifrado como firma digital en un solo protocolo, lo que lo hace versátil en una variedad de aplicaciones sin embargo para nuestro ejemplo resulto tener resultados un poco más bajos en comparación a otros algoritmos lo cual es correcto afirmar que puede ser más lento que otros algoritmos, especialmente en comparación con los otros resultados arrojados en ejecución. Whirlpool, una función hash robusta, ofrece alta seguridad contra colisiones y ataques criptográficos, como este también presenta dificultades en su aplicación, pero su rendimiento fue mejor que algunos de otros algoritmos como el ELGAMAL, aunque pudo ser complejo, es mejor en resultados.

RIPEMD es relativamente fácil de implementar y eficiente en términos de velocidad diferente a los anteriores algoritmos, pero con buenos resultados, pero puede ser menos resistente a ataques comparado con funciones hash más modernas.

CONCLUSIONES:

El peor candidato para este escenario podría ser RC4 debido a sus vulnerabilidades conocidas y su seguridad comprometida. Aunque es rápido de implementar y tiene una eficiencia computacional alta, ya que fue uno de los algoritmos que mejores resultados nos presento.

Por otro lado, el mejor candidato podría ser Blowfish o RIPEMD. Como estos algoritmos ofrecen un buen equilibrio entre seguridad y eficiencia, incluso en su implementación, lo cual es favorable en comparación de los demás algoritmos excepto RC4, lo que lo hace versátil en varias aplicaciones concretamente en nuestro ejemplo funcionaron bastante bien con buenos resultados.

Aunque su velocidad puede ser superada por algoritmos más modernos, o en este caso comprado a los otros 4 algoritmos más, como se mencionó anteriormente es importante también tomar en cuenta aspectos de seguridad y resistencia a los ataques lo convierten en una de las mejores opciones dentro de los 6 algoritmos que se analizaron.

BIBLIOGRAFIA:

[1] A. P. U. Siahaan, "An Overview of the RC4 Algorithm", 22-Sep-2017. [Online]. Available: osf.io/preprints/inarxiv/svufd.

[2] R. Rahim, "Combination of the Blowfish and Lempel-Ziv-Welch Algorithms for Text Compression", 04-Nov-2017. [Online]. Available: osf.io/preprints/inarxiv/c3qud.

[3] P. Hoffman, "Elliptic Curve Digital Signature Algorithm (DSA) for DNSSEC,"Apr. 2012. doi: 10.17487/rfc6605.

[4] O. A. Imran, S. F. Yousif, I. S. Hameed, W. N. A.-D. Abed, and A. T. Hammid, "Implementation of El-Gamal algorithm for speech signals encryption and decryption," Procedia Computer Science, vol. 167, pp. 1028–1037, Jan. 2020, doi: 10.1016/j.procs.2020.03.402.

[5] P. Kitsos and O. Koufopavlou, "Efficient architecture and hardware implementation of the Whirlpool hash function," in IEEE Transactions on Consumer Electronics, vol. 50, no. 1, pp. 208-213, Feb. 2004, doi: 10.1109/TCE.2004.1277864.