

Universidad Central del Ecuador
Facultad de Ingeniería y Ciencias Aplicadas
Carrera de Ingeniería en Ciencias de la Computación



Criptografía y seguridad de la información

Algoritmos Criptográficos DES, DSA, SHA-1

Integrantes:

- Altamirano Ortiz Jonathan Danilo
- Flórez Rivera Yaniry Mabely
- Gualoto Tigrero Erika Paola
- Simbaña Pulupa Pablo Fernando

Semestre: 8 vo

Fecha: 18/Noviembre/2024

Quito-2024



DES:

El **Data Encryption Standard (DES)** es un algoritmo de cifrado simétrico desarrollado en la década de 1970. Originalmente adoptado como un estándar de cifrado por el gobierno de los Estados Unidos, DES utiliza una clave de 56 bits para realizar operaciones de cifrado y descifrado en bloques de 64 bits. Funciona mediante una serie de permutaciones y sustituciones a través de 16 rondas de procesamiento. Aunque fue un método popular, la longitud relativamente corta de su clave lo hizo vulnerable a ataques de fuerza bruta y fue reemplazada en 2001 por el Advanced Encryption Standard (AES) en aplicaciones de alta seguridad.

```
public static void main(String[] args) throws Exception {
    // Medir el tiempo de generación de la clave
    long startKeyGenTime = System.nanoTime();
    // Generar clave de 56 bits para DES
    KeyGenerator keyGen = KeyGenerator.getInstance("DES");
    keyGen.init(56);
    SecretKey secretKey = keyGen.generateKey();
    long endKeyGenTime = System.nanoTime();

    // Mostrar la clave generada
    String encodedKey = Base64.getEncoder().encodeToString(secretKey.getEncoded());
    System.out.println("Clave generada: " + encodedKey);

    // Mostrar el tiempo que demoró en generar la clave en segundos
    System.out.println("Tiempo de generación de clave: " + ((endKeyGenTime - startKeyGenTime) / 1_000_000_000.0) + " segun");

    // Crear el objeto Cipher para cifrar con DES
    Cipher cipher = Cipher.getInstance("DES/ECB/PKCS5Padding");

    // Especificar la ruta del archivo
    Path filePath = Paths.get("data/palabras_10.txt");

    // Obtener el nombre del archivo
    String fileName = filePath.getFileName().toString();

    // Leer el archivo
    byte[] data = Files.readAllBytes(filePath);

    // Cifrar el archivo
    cipher.init(Cipher.ENCRYPT_MODE, secretKey);
    byte[] encryptedData = cipher.doFinal(data);

    // Guardar el archivo cifrado
    Path encryptedFilePath = Paths.get("data/palabras_10_cifrado.txt");
    Files.write(encryptedFilePath, encryptedData);

    // Mostrar el tiempo de cifrado en segundos
    long startCipherTime = System.nanoTime();
    // Cifrar el archivo
    cipher.init(Cipher.ENCRYPT_MODE, secretKey);
    byte[] encryptedData = cipher.doFinal(data);
    long endCipherTime = System.nanoTime();

    // Mostrar el tiempo de cifrado en segundos
    System.out.println("Tiempo de cifrado: " + ((endCipherTime - startCipherTime) / 1_000_000_000.0) + " segun");

    // Leer el archivo cifrado
    byte[] encryptedData2 = Files.readAllBytes(encryptedFilePath);

    // Descifrar el archivo
    cipher.init(Cipher.DECRYPT_MODE, secretKey);
    byte[] decryptedData = cipher.doFinal(encryptedData2);

    // Guardar el archivo descifrado
    Path decryptedFilePath = Paths.get("data/palabras_10_descifrado.txt");
    Files.write(decryptedFilePath, decryptedData);

    // Mostrar el tiempo de descifrado en segundos
    long startDecryptTime = System.nanoTime();
    // Descifrar el archivo
    cipher.init(Cipher.DECRYPT_MODE, secretKey);
    byte[] decryptedData = cipher.doFinal(encryptedData2);
    long endDecryptTime = System.nanoTime();

    // Mostrar el tiempo de descifrado en segundos
    System.out.println("Tiempo de descifrado: " + ((endDecryptTime - startDecryptTime) / 1_000_000_000.0) + " segun");

    // Mostrar los resultados
    System.out.println("Clave generada: " + encodedKey);
    System.out.println("Tiempo de generación de clave: " + ((endKeyGenTime - startKeyGenTime) / 1_000_000_000.0) + " segun");
    System.out.println("Tiempo de cifrado: " + ((endCipherTime - startCipherTime) / 1_000_000_000.0) + " segun");
    System.out.println("Tiempo de descifrado: " + ((endDecryptTime - startDecryptTime) / 1_000_000_000.0) + " segun");
    System.out.println("Número de caracteres de entrada: " + data.length);
    System.out.println("Número de caracteres de salida: " + decryptedData.length);
    System.out.println("Tiempo total de ejecución: " + ((endDecryptTime - startKeyGenTime) / 1_000_000_000.0) + " segun");
}
```

Problems | Javadoc | Declaration | Console x

<terminated> algoritmoDES (1) (Java Application) C:\Program Files\Java\jdk-22\bin\javaw.exe (18 nov 2024, 8:38:33 p. m. - 8:38:35 p. m.) [pid: 11620]

Clave generada: d4e4vQ7P=

Tiempo de generación de clave: 0.1620726 segundos

Leyendo archivo: palabras_10.txt

Texto cifrado: A0X1Mu0d08y9j0t1bDuo0F0904p0q0v0A03v29veYX+Q0g27f13Vtc7DjaW0utSvrA=

Texto descifrado: Una suave brisa mueve las hojas de los árboles altos.

RESULTADOS

Tiempo de cifrado: 5.165E-4 segundos

Tiempo de descifrado: 1.799E-4 segundos

Tiempo de lectura del archivo: 0.0055592 segundos

Número de caracteres de entrada: 53

Número de caracteres de salida: 76

Tiempo total de ejecución: 0.1813848 segundos

```
public static void main(String[] args) throws Exception {
    // Medir el tiempo de generación de la clave
    long startKeyGenTime = System.nanoTime();
    // Generar clave de 56 bits para DES
    KeyGenerator keyGen = KeyGenerator.getInstance("DES");
    keyGen.init(56);
    SecretKey secretKey = keyGen.generateKey();
    long endKeyGenTime = System.nanoTime();

    // Mostrar la clave generada
    String encodedKey = Base64.getEncoder().encodeToString(secretKey.getEncoded());
    System.out.println("Clave generada: " + encodedKey);

    // Mostrar el tiempo que demoró en generar la clave en segundos
    System.out.println("Tiempo de generación de clave: " + ((endKeyGenTime - startKeyGenTime) / 1_000_000_000.0) + " segun");

    // Crear el objeto Cipher para cifrar con DES
    Cipher cipher = Cipher.getInstance("DES/ECB/PKCS5Padding");

    // Especificar la ruta del archivo
    Path filePath = Paths.get("data/palabras_1000000.txt");

    // Obtener el nombre del archivo
    String fileName = filePath.getFileName().toString();

    // Leer el archivo
    byte[] data = Files.readAllBytes(filePath);

    // Cifrar el archivo
    cipher.init(Cipher.ENCRYPT_MODE, secretKey);
    byte[] encryptedData = cipher.doFinal(data);

    // Guardar el archivo cifrado
    Path encryptedFilePath = Paths.get("data/palabras_1000000_cifrado.txt");
    Files.write(encryptedFilePath, encryptedData);

    // Mostrar el tiempo de cifrado en segundos
    long startCipherTime = System.nanoTime();
    // Cifrar el archivo
    cipher.init(Cipher.ENCRYPT_MODE, secretKey);
    byte[] encryptedData = cipher.doFinal(data);
    long endCipherTime = System.nanoTime();

    // Mostrar el tiempo de cifrado en segundos
    System.out.println("Tiempo de cifrado: " + ((endCipherTime - startCipherTime) / 1_000_000_000.0) + " segun");

    // Leer el archivo cifrado
    byte[] encryptedData2 = Files.readAllBytes(encryptedFilePath);

    // Descifrar el archivo
    cipher.init(Cipher.DECRYPT_MODE, secretKey);
    byte[] decryptedData = cipher.doFinal(encryptedData2);

    // Guardar el archivo descifrado
    Path decryptedFilePath = Paths.get("data/palabras_1000000_descifrado.txt");
    Files.write(decryptedFilePath, decryptedData);

    // Mostrar el tiempo de descifrado en segundos
    long startDecryptTime = System.nanoTime();
    // Descifrar el archivo
    cipher.init(Cipher.DECRYPT_MODE, secretKey);
    byte[] decryptedData = cipher.doFinal(encryptedData2);
    long endDecryptTime = System.nanoTime();

    // Mostrar el tiempo de descifrado en segundos
    System.out.println("Tiempo de descifrado: " + ((endDecryptTime - startDecryptTime) / 1_000_000_000.0) + " segun");

    // Mostrar los resultados
    System.out.println("Clave generada: " + encodedKey);
    System.out.println("Tiempo de generación de clave: " + ((endKeyGenTime - startKeyGenTime) / 1_000_000_000.0) + " segun");
    System.out.println("Tiempo de cifrado: " + ((endCipherTime - startCipherTime) / 1_000_000_000.0) + " segun");
    System.out.println("Tiempo de descifrado: " + ((endDecryptTime - startDecryptTime) / 1_000_000_000.0) + " segun");
    System.out.println("Número de caracteres de entrada: " + data.length);
    System.out.println("Número de caracteres de salida: " + decryptedData.length);
    System.out.println("Tiempo total de ejecución: " + ((endDecryptTime - startKeyGenTime) / 1_000_000_000.0) + " segun");
}
```

Problems | Javadoc | Declaration | Console x

<terminated> algoritmoDES (1) (Java Application) C:\Program Files\Java\jdk-22\bin\javaw.exe (18 nov 2024, 8:51:12 p. m. - 8:53:30 p. m.) [pid: 27816]

Texto descifrado: Los gatos se estiran perezosamente bajo el sol cálido. La ciudad despierta lentamente mientras las luces se encienden. La luna ilumina la noche mientras las

RESULTADOS

Tiempo de cifrado: 0.9790492 segundos

Tiempo de descifrado: 1.2929182 segundos

Tiempo de lectura del archivo: 136.5572567 segundos

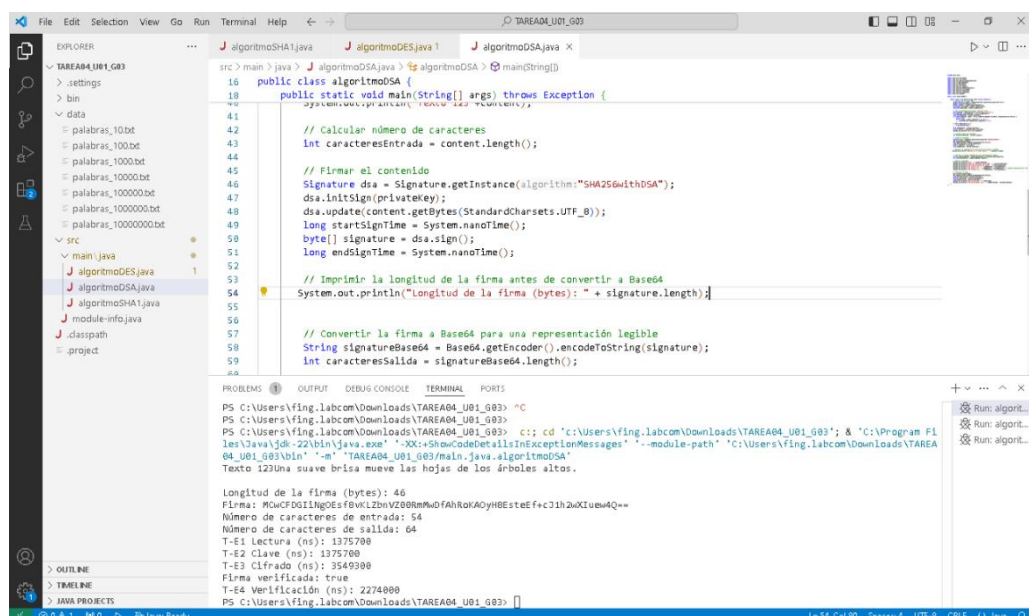
Número de caracteres de entrada: 61884084

Número de caracteres de salida: 83630844

Tiempo total de ejecución: 136.705433 segundos

DSA:

El **Algoritmo de Firma Digital (DSA)** es un algoritmo de firma digital desarrollado en 1991 por el NIST para autenticar documentos digitales. Este algoritmo de criptografía asimétrica utiliza una clave privada para firmar el mensaje y una clave pública para verificar la firma, proporcionando integridad y autenticidad sin la necesidad de cifrar el mensaje completo. DSA es parte del estándar DSS (Digital Signature Standard) y sigue siendo usado en diversas aplicaciones donde la autenticidad es crítica, aunque se han desarrollado variantes más modernas como ECDSA (Elliptic Curve DSA).

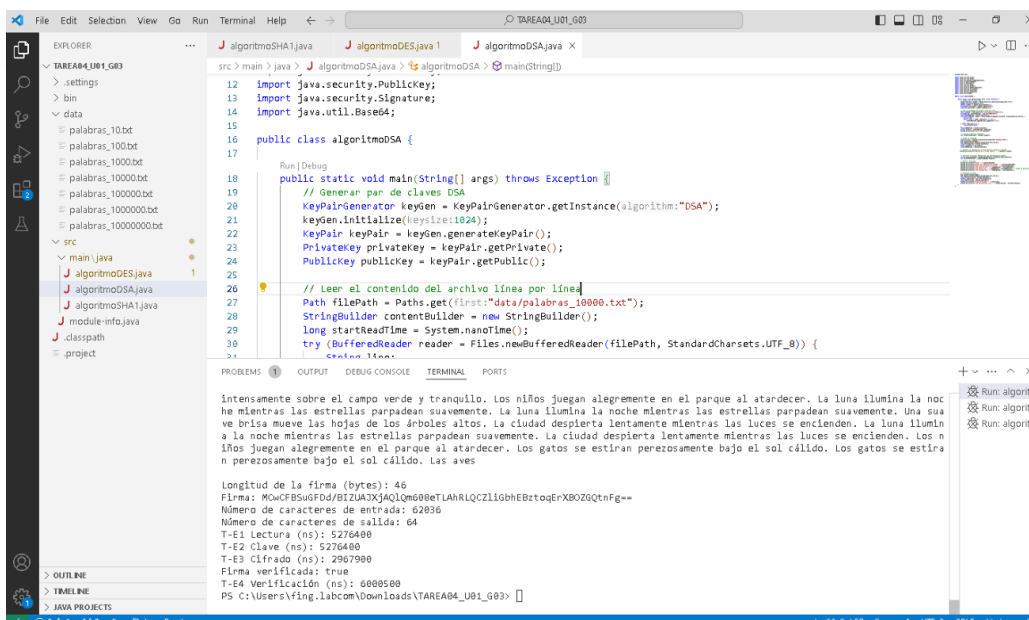


```
src > main > java -J algoritmoDSA.java -J algoritmoDSA > main(String[] args) throws Exception {
16 public class algoritmoDSA {
17     public static void main(String[] args) throws Exception {
18         // Calcular número de caracteres
19         int caracteresEntrada = content.length();
20
21         // Firmar el contenido
22         Signature dsas = Signature.getInstance(algorithm:"SHA256withDSA");
23         dsas.initSign(privateKey);
24         dsas.update(content.getBytes(StandardCharsets.UTF_8));
25         long startSignTime = System.nanoTime();
26         byte[] signature = dsas.sign();
27         long endSignTime = System.nanoTime();
28
29         // Imprimir la longitud de la firma antes de convertir a Base64
30         System.out.println("Longitud de la firma (bytes): " + signature.length);
31
32         // Convertir la firma a Base64 para una representación legible
33         String signatureBase64 = Base64.getEncoder().encodeToString(signature);
34         int caracteresSalida = signatureBase64.length();
35     }
36 }
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\Fing.Labcom\Downloads\TAREA04_U01_G03> "C
PS C:\Users\Fing.Labcom\Downloads\TAREA04_U01_G03>
PS C:\Users\Fing.Labcom\Downloads\TAREA04_U01_G03> cd 'c:\Users\Fing.Labcom\Downloads\TAREA04_U01_G03'; & 'C:\Program Files\Java\jdk-22\bin\java.exe' '-XX:+ShowCodeDetailsInExceptionMessages' '-module-path' 'C:\Users\Fing.Labcom\Downloads\TAREA04_U01_G03\bin' '-m' 'TAREA04_U01_G03\main.java.algoritmoDSA'
Texto 123Una suave brisa mueve las hojas de los árboles altos.

Longitud de la firma (bytes): 46
Firma: M0wCFDGIInQ0esf8VKLZbnVZ08RM0FAH0kAOyHBEsteEfc31h2uXIuw4Q==
Número de caracteres de entrada: 54
Número de caracteres de salida: 64
T-E1 Lectura (ns): 1375700
T-E2 Clave (ns): 1375700
T-E3 Cifrado (ns): 3549900
Firma verificada: true
T-E4 Verificación (ns): 2274000
PS C:\Users\Fing.Labcom\Downloads\TAREA04_U01_G03>
```



```
src > main > java -J algoritmoDSA.java -J algoritmoDSA > main(String[] args) throws Exception {
12 import java.security.PublicKey;
13 import java.security.Signature;
14 import java.util.Base64;
15
16 public class algoritmoDSA {
17     // Generar par de claves DSA
18     KeyPairGenerator keyGen = KeyPairGenerator.getInstance(algorithm:"DSA");
19     keyGen.initialize(1024);
20     KeyPair keyPair = keyGen.generateKeyPair();
21     PrivateKey privateKey = keyPair.getPrivate();
22     PublicKey publicKey = keyPair.getPublic();
23
24     // Leer el contenido del archivo línea por línea
25     Path filePath = Paths.get("data/palabras_10000.txt");
26     StringBuilder contentBuilder = new StringBuilder();
27     long startReadTime = System.nanoTime();
28     try (BufferedReader reader = Files.newBufferedReader(filePath, StandardCharsets.UTF_8)) {
29         contentBuilder.append(reader.readLine());
30     }
31 }
```

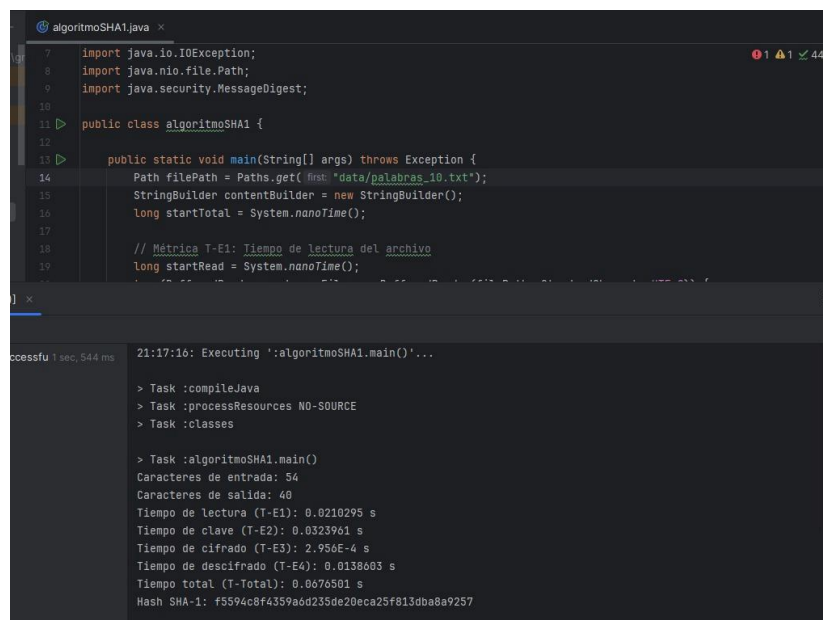
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
Intensamente sobre el campo verde y tranquilo. Los niños juegan alegremente en el parque al atardecer. La luna ilumina la noche mientras las estrellas parpadean suavemente. La luna ilumina la noche mientras las estrellas parpadean suavemente. Una suave brisa mueve las hojas de los árboles altos. La ciudad despierta lentamente mientras las luces se encienden. La luna ilumina la noche mientras las estrellas parpadean suavemente. La ciudad despierta lentamente mientras las luces se encienden. Los niños juegan alegremente en el parque al atardecer. Los gatos se estiran perezosamente bajo el sol cálido. Los gatos se estiran perezosamente bajo el sol cálido. Las aves

Longitud de la firma (bytes): 46
Firma: M0wCFB5uGF0d/8tZU3Xj9aQl0m600EtLAHRLQCZl1GbHEBztoqErXB0ZG0tnFg==
Número de caracteres de entrada: 62936
Número de caracteres de salida: 64
T-E1 Lectura (ns): 5276400
T-E2 Clave (ns): 5276400
T-E3 Cifrado (ns): 2967900
Firma verificada: true
T-E4 Verificación (ns): 6000500
PS C:\Users\Fing.Labcom\Downloads\TAREA04_U01_G03>
```

SHA-1:

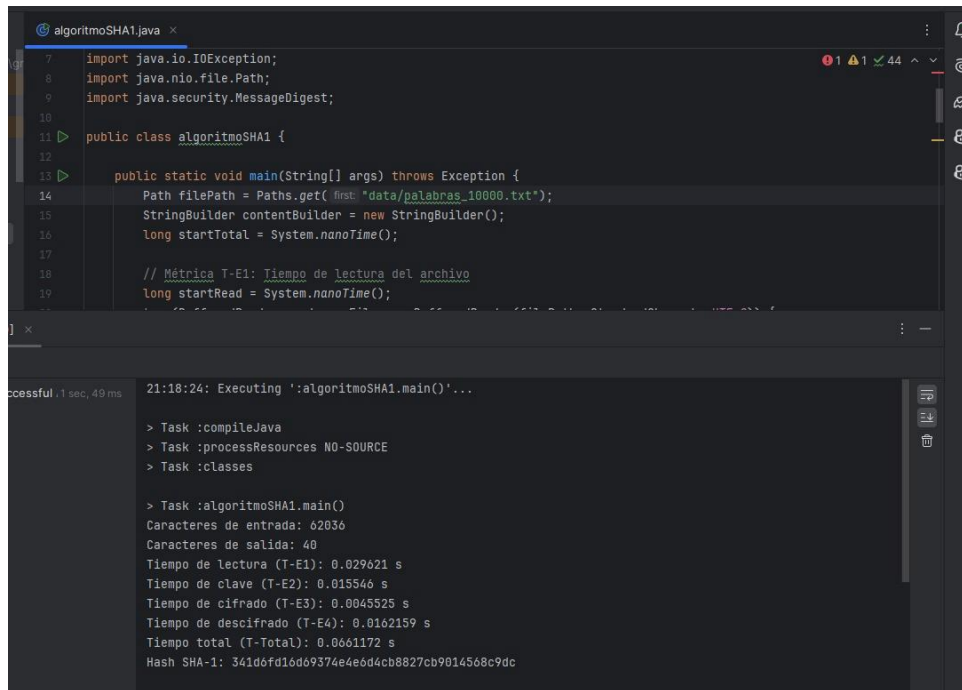
El **Secure Hash Algorithm 1 (SHA-1)** es un algoritmo de hash criptográfico desarrollado por la NSA en 1993. SHA-1 produce un resumen o "hash" de 160 bits a partir de una entrada de cualquier longitud. Inicialmente fue utilizado ampliamente para verificar la integridad de datos y autenticación de mensajes. Sin embargo, con el tiempo, se descubrieron debilidades en SHA-1 que permitían ataques de colisión, donde dos mensajes distintos pueden generar el mismo hash. Esto ha llevado a la mayoría de las organizaciones a migrar a algoritmos más seguros, como SHA-256 y SHA-3.



```
algoritmoSHA1.java x
7  import java.io.IOException;
8  import java.nio.file.Path;
9  import java.security.MessageDigest;
10
11 public class algoritmoSHA1 {
12
13     public static void main(String[] args) throws Exception {
14         Path filePath = Paths.get("data/palabras_10.txt");
15         StringBuilder contentBuilder = new StringBuilder();
16         long startTotal = System.nanoTime();
17
18         // Métrica T-E1: Tiempo de lectura del archivo
19         long startRead = System.nanoTime();
20         // ... (código omitido) ...
21     }
22 }

21:17:16: Executing 'algoritmoSHA1.main()'...
> Task :compileJava
> Task :processResources NO-SOURCE
> Task :classes

> Task :algoritmoSHA1.main()
Caracteres de entrada: 54
Caracteres de salida: 40
Tiempo de lectura (T-E1): 0.0210295 s
Tiempo de clave (T-E2): 0.0323961 s
Tiempo de cifrado (T-E3): 2.956E-4 s
Tiempo de descifrado (T-E4): 0.0138603 s
Tiempo total (T-Total): 0.0676501 s
Hash SHA-1: f5594c8f4359a6d235de20eca25f813dba8a9257
```



```
algoritmoSHA1.java x
7 import java.io.IOException;
8 import java.nio.file.Path;
9 import java.security.MessageDigest;
10
11 public class algoritmoSHA1 {
12
13     public static void main(String[] args) throws Exception {
14         Path filePath = Paths.get("data/palabras_10000.txt");
15         StringBuilder contentBuilder = new StringBuilder();
16         long startTotal = System.nanoTime();
17
18         // Métrica T-E1: Tiempo de lectura del archivo
19         long startRead = System.nanoTime();
20
21         // ... (rest of the code is partially visible)
22     }
23 }

21:18:24: Executing 'algoritmoSHA1.main()'...
> Task :compileJava
> Task :processResources NO-SOURCE
> Task :classes

> Task :algoritmoSHA1.main()
Caracteres de entrada: 62036
Caracteres de salida: 40
Tiempo de lectura (T-E1): 0.029621 s
Tiempo de clave (T-E2): 0.015546 s
Tiempo de cifrado (T-E3): 0.0045525 s
Tiempo de descifrado (T-E4): 0.0162159 s
Tiempo total (T-Total): 0.0661172 s
Hash SHA-1: 341d6fd10d69374e4e6d4cb8827cb9814568c9dc
```

BIBLIOGRAFÍA:

[1] Instituto Nacional de Estándares y Tecnología (NIST), "Estándar de hash seguro (SHS)", Publicación de SHS del NIST Departamento de Defensa de los Estados Unidos <https://nvlp.norte.gramo/norte/FIP/NORTE.FIPS.180-4.pdf>.

[2] J. Kelsey y B. Schneier, "Segundas preimágenes en funciones hash de n bits para un trabajo mucho menor que 2^n ", en Taller Internacional sobre Cifrado Rápido de Software.

[3] Instituto Nacional de Estándares y Tecnología (NIST), "Estándar de firma digital (DSS)", Publicación DSS del NIST. <https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-4.pdf>.

[4] B. Schneier, Criptografía aplicada: protocolos, algoritmos y código fuente en C, 2.^a ed., Nueva York: Wiley, 1995, págs. 485-487.

ANEXOS:

- Tabla de resultados (T-1, T-2, T-3, T-4 y T-Total):

	#palabras	#caracteres_entrada	#caracteres_salida	T-E1 Lectura (ns)	T-E2 Clave (ns)	T-E3 Cifrado (ns)	T-E4 Descifrado(ns)	T-Total(ns)	T-Total(ms)	T-Total(s)
DES	10	53	76	5647200	67333800	302900	296300	73580200	73,5802	0,0735802
	100	608	832	5631400	63725700	527400	445200	70329700	70,3297	0,0703297
	1000	6088	8248	7579900	62363800	1269900	626300	71839900	71,8399	0,0718399
	10000	62035	62035	20813600	75882000	5304400	4795800	106795800	106,7958	0,1067958
	100000	618748	836184	70802400	69743600	21015400	20923100	182484500	182,4845	0,1824845
	1000000	6185049	8359436	964034600	445375700	352053800	316569500	2078033600	2078,0336	2,0780336
	10000000	61884084	83636044	8423898700	491995800	2123690900	4581400500	15620985900	15620,9859	15,6209859
DSA	10	53	64	28239701	28239701	48286346	0	104765748	104,765748	1,04766E-07
	100	608	64	18555927	18555927	7888080	0	44999934	44,999934	4,49999E-08
	1000	6088	64	31694505	31694505	11322939	0	74711949	74,711949	7,47119E-08
	10000	62035	64	22867846	22867846	9166593	0	54902285	54,902285	5,49023E-08
	100000	618748	64	60385856	60385856	7505760	0	128277472	128,277472	1,28277E-07
	1000000	6185049	64	217509419	217509419	6869952	0	441888790	441,88879	4,41889E-07
	10000000	61884084	64	927091452	927091452	14636449	0	1868819353	1868,819353	1,86882E-06
SHA-1	10	53	40	23026700	11383600	9191800	0	54568500	54,5685	5,45685E-08
	100	608	40	13251600	13232800	8594200	0	45897000	45,897	4,5897E-08
	1000	6088	40	14615600	13872200	62533500	0	112868000	112,868	1,12868E-07
	10000	62035	40	7464100	12891100	8557900	0	44811800	44,8118	4,48118E-08
	100000	618748	40	22726400	24001500	46476200	0	124019400	124,0194	1,24019E-07
	1000000	6185049	40	7930100	12278200	18425400	0	60043700	60,0437	6,00437E-08
	10000000	61884084	40	6037800	10022900	8806100	0	35641100	35,6411	3,56411E-08