PONTIFICIA UNIVERSIDAD CATÓLICA DE VALPARAÍSO

Escuela de Ingeniería Eléctrica
PONTIFICIA UNIVERSIDAD CATOLICA DE VALPARAISO

Matías Ignacio Hernández Orrego

Implementación de un generador y verificador del código de seguridad MD5

Informe Proyecto de Título de Ingeniero Electrónico

Escuela de Ingeniería Eléctrica
Facultad de Ingeniería

Valparaíso, 03 de julio de 2019

Implementación de un generador y verificador del código de seguridad MD5

Matías Ignacio Hernández Orrego

Informe Final para optar al título de Ingeniero Electrónico,

aprobada por la comisión de la

Escuela de Ingeniería Eléctrica de la

Facultad de Ingeniería de la

Pontificia Universidad Católica de Valparaíso

conformada por

Sr. David Alberto Velasco López

Profesor Guía

Sr. Daniel Alberto Gutierrez Dondero

Segundo Revisor

Sr. Sebastian Carlos Fingerhuth Massmann

Secretario Académico

Valparaíso, 03 de julio de 2019

Agradecimientos

Quiero agradecer a mi familia por su apoyo incondicional en todos estos años de formación, en especial a mi madre Sandra, a mi abuela Georgina, y a todas las personas que estuvieron apoyándome en los momentos difíciles, me ayudaron a no decaer nunca y seguir adelante en este complejo mundo que es la universidad. Por todo su apoyo muchas gracias.

Valparaíso, 5 de marzo de 2019

Resumen

Durante los meses de marzo a diciembre se realizó un amplio estudio del algoritmo de seguridad MD5. Se hizo una introducción en la materia y en las competencias básicas necesarias para desarrollarlo. Se estudiaron, además de su desarrollo histórico, los métodos criptográficos y de función HASH anteriores como fueron los MD2 y MD4.

Se darán ciertas propuestas para desarrollar el código generador y verificador del algoritmo MD5 para empezar nuestro trabajo sobre códigos que puedan ayudar a llevarlo al código deseado.

Se seguirá con el estudio del estado del arte de los algoritmos SHA, su evolución, aplicaciones y seguridades que nos brinda hasta el día de hoy en el ámbito de la criptografía.

Se presentará el lenguaje virtual Pascal, definido en los objetivos anteriores, para desarrollar las principales estructuras a utilizar en la programación del algoritmo. Se conocerá su forma de programar, sus estructuras y se explicará paso a paso, hasta concluir con la programación del programa que se utilizará finalmente en la implementación de verificación de semillas arbitrarias.

Una vez terminada la programación, se harán pruebas que permitan verificar que el programa no presenta problemas y la encriptación sea la correcta. Se harán comparaciones entre semillas arbitrarias para verificar que el largo de 128 bits sea el correcto para el resumen del HASH para MD5.

Luego se puede dejar explícito las opciones que tiene el lenguaje de programación virtual Pascal y su compilador algunos de estos menús se muestran solo con fines pedagógicos para el lector que no esté instruido en el lenguaje de programación, y otros son de uso vital para la comprobación del correcto funcionamiento. Terminando con el trabajo en la programación del algoritmo MD5, enfocado principalmente en la transmisión de datos, los cuales puedes ser transmitidos byte a byte.

Palabras claves: HASH, Algoritmo, encriptación, byte, bits, Colisión, Criptografía, Seguridad, Protocolo, Vulnerabilidad, Cifrado, lenguaje, Programación, tiempo, Pseudocódigo, Transferencia, Round.

Abstract

During the months of March to December, an extensive study of the MD5 security algorithm was carried out. An introduction was made in the subject and in the basic competences necessary to develop it. In addition to its historical development, the previous cryptographic and HASH function methods were studied, such as MD2 and MD4.

On the other hand, in the thesis reference will be made especially to the different applications that the MD5 algorithm has in the computer world, and how important it is in the field of computer security, and this, in the security of the market.

Certain proposals will be given to develop the code generator and verifier of the MD5 algorithm to begin our work on codes that can help bring it to the desired code.

It will continue with the study of the state of the art of the SHA algorithms, their evolution, applications and security that gives us to this day in the field of cryptography.

The Pascal virtual language, defined in the previous objectives, will be presented to develop the main structures to be used in the programming of the algorithm. We will know how to program, its structures and will be explained step by step, until we conclude with the programming of the program that will be used finally in the implementation of verification of arbitrary seeds.

Once the programming is finished, tests will be done to verify that the program does not present problems and the encryption is correct. Comparisons between arbitrary seeds will be made to verify that the length of 128 bits is correct for the HASH summary for MD5.

Then you can leave explicit the options that the virtual programming language Pascal and its compiler have, some of these menus are shown only for pedagogical purposes for the reader who is not instructed in the programming language, and others are of vital use for checking of the correct functioning. Finishing with the work in the programming of the MD5 algorithm, focused mainly on data transmission, which can be transmitted byte to byte.

Keywords: HASH, Algorithm, Encryption, Byte, Bit, Collision, Cryptography, Security, Protocol, Vulnerability, Encryption, Language, Programming, Time, Pseudocode, Transfer, Round.

Índice general

Introducción

Hoy en día nos encontramos en una era informática donde se hace cada vez más necesario cuidar nuestra información, y al mismo tiempo ser capaces garantizar la satisfacción de los usuarios sobre el contenido de lo vuelca en los sistemas o redes informáticas. Es por esto por lo que surgen los dispositivos de seguridad que proveen protección contra las amenazas externas al traspasar información, siendo los métodos criptográficos la medida más confiable para asegurarnos que la información que se manipula está protegida. Esto nos permite evitar la interrupción, modificación, interceptación y usurpación.

Es aquí donde nacen las funciones HASH que son algoritmos de encriptación de la información digital. Estos han sido desarrollados en conjunto con el incremento de las necesidades del usuario en cuanto a los riesgos que se han presentado en la red, que van haciendo cada día más vulnerable las medidas de seguridad.

Uno de los principales algoritmos de encriptación del tipo HASH es el algoritmo MD5, que reúne todas las condiciones para asegurar un encriptado exitoso, confidencialidad, integridad, autenticación y no repudio entre emisor y receptor.

Para asegurar la confidencialidad y el resguardo de la información ante cualquier factor no deseado, se hace necesario indagar más allá de los métodos como el MD5. El estudio del algoritmo SHA se hace fundamental para nutrir el proyecto y mostrar la evolución de este, y como sus actualizaciones han ido superando barreras de vulnerabilidad hasta demostrar seguridad ante cualquier ataque.

Por otro lado, para ejecutar el algoritmo MD5 son necesarias una serie de reglas específicas en su estructura, que nos permitirán señalar de manera clara y ordenada la acción que se quiere ejecutar dentro del algoritmo. Eso nos lleva a la toma de decisiones sobre los lenguajes de programación que serán utilizados y los motivos de su uso.

Un lenguaje de programación informático se define como cualquier estructura que permite entregar una secuencia de instrucciones para el correcto procesamiento de datos por parte de la computadora. Se entiende que la traducción del lenguaje humano, a la computadora, debe ser sistemático en sus instrucciones. Y es importante hacer notar que cada una de sus opciones nos permite trabajar de una manera expedita para el usuario.

Uno de estos lenguajes o estructura de instrucciones, para desarrollar el algoritmo MD5 fue el lenguaje Virtual Pascal. Cuyo autor original es Vitaly Miryanov quien lo publicó en 1995, y más tarde desarrollado por Allan Mertner.

Así, a través del estudio de cada uno de sus elementos, errores y aciertos, más allá de centrarnos en el algoritmo MD5, comprenderemos el desarrollo lógico del programa, su estructura y la compilación de información necesaria hasta convertirlo en un código final, de prueba y depuración del programa.

Finalmente haremos una síntesis de la transmisión de datos donde es útil el MD5. Sistemas donde es necesario buscar alternativas seguras de entrega, protocolos legales y formas que son necesarias para que el traspaso de datos sea seguro. Se explicarán de modo que el lector tenga la mayor información para la toma de decisiones que le pueda garantizar el éxito en la transferencia de datos con MD5.

También, se someterá a MD5 a pruebas para certificar la conexión de manera segura entre los usuarios de una red más allá de solo encriptar una semilla arbitraria, sino también en la transmisión de archivo en su totalidad a través del algoritmo, estudiando enlaces para hacer más factible la conexión.

Y finalmente haremos una evaluación del costo económico de la encriptación con MD5, considerando que, dependiendo de la dificultad de la entrada, el mayor costo es el factor tiempo, el cual debe ser conocido.

Objetivos generales

- Desarrollar la generación de una palabra de autentificación a partir de una semilla arbitraria. Se orienta el trabajo a la validación de usuarios y contraseñas en el entorno redes de computadores.

Objetivos especificos

- Estudio del estado del arte del algoritmo de seguridad en redes de computadores haciendo énfasis en el algoritmo MD5.
- Implementar la generación de contraseñas a partir de entradas arbitrarias para el algoritmo de seguridad MD5.

1 Algoritmo

En el contexto matemático, los algoritmos son una serie de normas o reglas específicas que hace posible la ejecución de actividades, cumpliendo una serie de pasos continuos que no originan duda a la persona que realiza dicha actividad. Los algoritmos se pueden expresar de diversas formas: lenguaje natural, diagrama de flujo, Pseudocódigo y finalmente, lenguaje de programación [1].

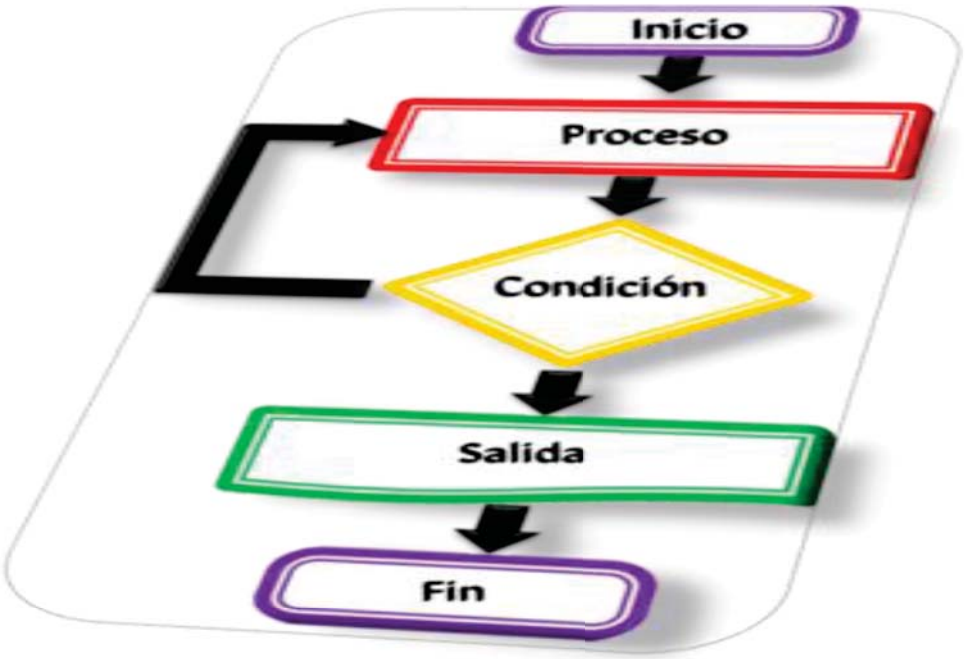


Figura 1-1 : Representación de algoritmo (fuente: didacsoftware.com)

Los algoritmos tienen como características que definen su efectividad paso a paso, lo que quiere decir que una persona puede hacer un algoritmo sin utilizar un ordenador y los pasos de este son finitos. Cada acción contempla un inicio establecido y un final esperado. Sino se cumple, el flujo se interrumpe.

Hay algoritmos cualitativos, en que las órdenes vienen dadas a través de palabras [1].

Hay también algoritmos cuantitativos, que son aquellos que se expresan de manera matemática. Por ejemplo, el conjunto de instrucciones que permite, mediante una operación aritmética, conocer la raíz cuadrada de un número. Puede haber uno o más algoritmos para dicha operación.

También encontramos los algoritmos computacionales, que son los que utilizaremos en el desarrollo de este proyecto. Son un conjunto de instrucciones dadas a un programa contenido en un computador, para que el flujo de información que se le ingrese se procese de manera específica y el resultado provea una nueva información resultante o una acción determinada. Son una suma de órdenes alfanuméricas a través de las que podemos alcanzar un orden adecuado para el desarrollo de cualquier actividad y garantizar una correcta ejecución de esas tareas [1].

1.1 Criptografía

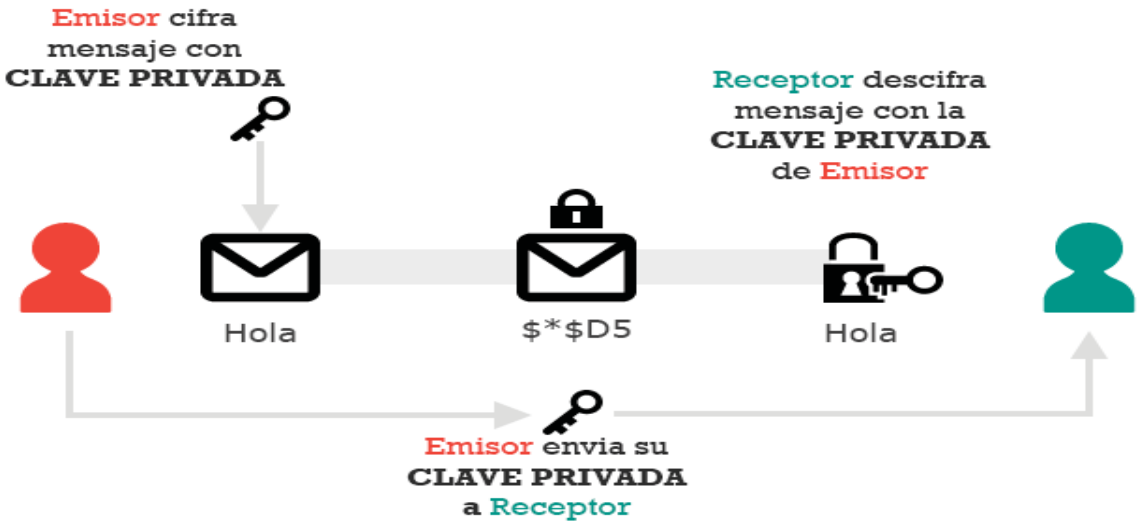
La criptografía es una técnica para ocultar información frente a observadores no autorizados dando protección sobre su modificación, interpretación y la inserción de información extra. En el área informática puede ser usada para prevenir el acceso y uso no autorizado de los datos de una red o sistema. Se usan esas técnicas al ingresar la información y luego al extraer la información del sistema, para volverla nuevamente íntegra.

Hoy en día la criptografía es la metodología que provee seguridad en las redes telemáticas, siendo utilizada en gran parte en la identificación y autentificación de identidades, controlando el acceso a recursos, manteniendo la confidencialidad e integridad de los mensajes durante su transporte entre los distintos terminales a donde se supone, debe llegar intacta.

1.1.1 Criptografía simétrica

Se basa principalmente en un sistema que ocupa la misma clave para cifrar y descifrar un mensaje, las dos partes que se comunica han de ponerse de acuerdo para el traspaso de la clave bajo la cual se cifrará el mensaje. Una vez que las dos partes estén en conocimiento de ésta, la parte remitente cifra el mensaje (encriptación) con la clave única y enviando la información cifrada al destinatario el cual a su vez descifrará el mensaje utilizando la misma clave con la que se encripto.

El principal problema de seguridad reside en el intercambio de la clave entre el emisor y receptor, ya que éstos deben buscar un canal de comunicación que sea seguro. También es importante que la clave sea difícil de deducir por el hecho de que hoy en día los ordenadores pueden descifrar claves con mucha facilidad y de forma rápida. A esto hay que sumarle que los algoritmos criptográficos son de uso público, por lo cual las prevenciones y resguardos deben estar centrados en la complejidad interna y longitud de la clave empleada, para evitar ataque que pueda romper



la encriptación.

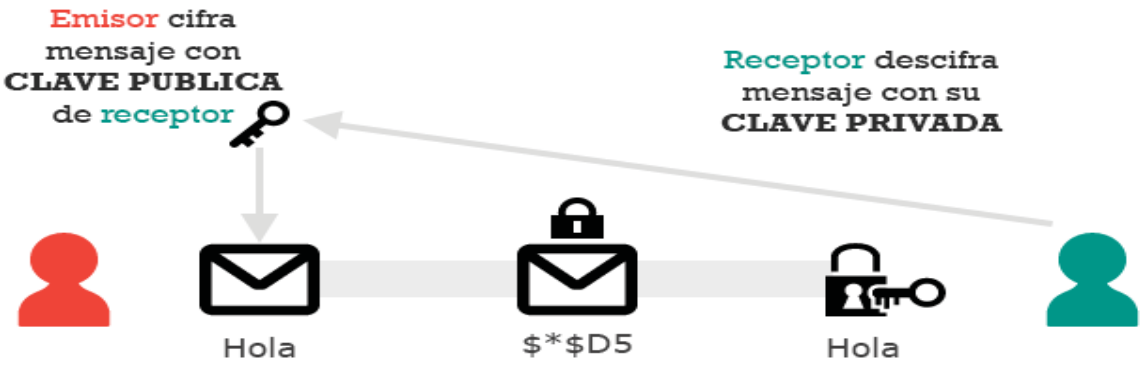
Figura 1-2 Ejemplo criptografía simétrica (fuente: es.paperblog.com)

1.1.2 Criptografia asimétrica

La criptografía asimétrica es un método de claves alternativo que consta de la generación de dos claves, una toma el nombre de clave pública y la otra, clave privada. La primera es de conocimiento público por lo cual es distribuida entre todas las personas que les incumbe el mensaje, y la clave privada debe ser guardada por el propietario de manera que nadie tenga acceso a ella. Además, el método asimétrico garantiza que esta pareja de claves será generada solamente una vez, lo que aseguraría que en ninguna circunstancia dos personas puedan obtener las mismas claves.

El método se basa principalmente que al momento de codificar un mensaje con la clave pública solo puede ser decodificado con la clave privada, lo que significa que al momento de codificar un mensaje con la clave pública solo el usuario dueño de la clave privada lo puede decodificar, haciendo el proceso más seguro ya que se resuelve el problema de envíos de claves.

Una de las desventajas de la criptografía asimétrica es que para la generación de claves se requiere más tiempo de proceso, ya que este proceso está basado en la utilización de un número compuesto de dos números primos muy grandes en la clave pública, con el cual el algoritmo cifra el mensaje y que solo la clave privada conoce esos factores primos con los cuales es muy rápido descifrar el mensaje, lo que hace que las claves deban ser de un largo considerable para hacer más difícil el proceso inverso el cual consta en la factorización de un número compuesto para obtener los números primos. Se recomienda de que la clave pública posea 1024 bits de largo como mínimo para que en el caso de ataque de fuerza tenga que ser factorizado para que el espectro de número



de factorizaciones sea demasiado grande.

Figura 1-3 Ejemplo criptografía asimétrica (Fuente: es.paperblog.com)

1.2 Funcion HASH

Los HASH o funciones de resumen son algoritmos que a partir de una entrada (texto, contraseña, imagen o un archivo entre otros), pueden crear una salida alfanumérica de longitud fija, es decir, independiente de la longitud que tenga la entrada, la salida siempre tendrá una longitud definida. Ésta representa un resumen de toda la información, con esto se puede decir que se crea una cadena de datos a la salida que solo se puede volver a crear introduciendo los mismos datos. Un ejemplo simple de cómo funciona un HASH lo vemos representado en la figura 1-4, donde en la primera entrada tenemos la palabra Fox la cual posee solo 3 letras, y a la salida tenemos la encriptación de esta palabra un código alfanumérico de largo definido de 40 caracteres. Si nos fijamos en la segunda entrada de la figura, el ingreso que tenemos es de un largo superior a la primera entrada, pero a la salida seguimos obteniendo un código de este largo, pero con distintas letras, ya que es imposible encontrar dos entradas distintas que posean la misma salida. [2]

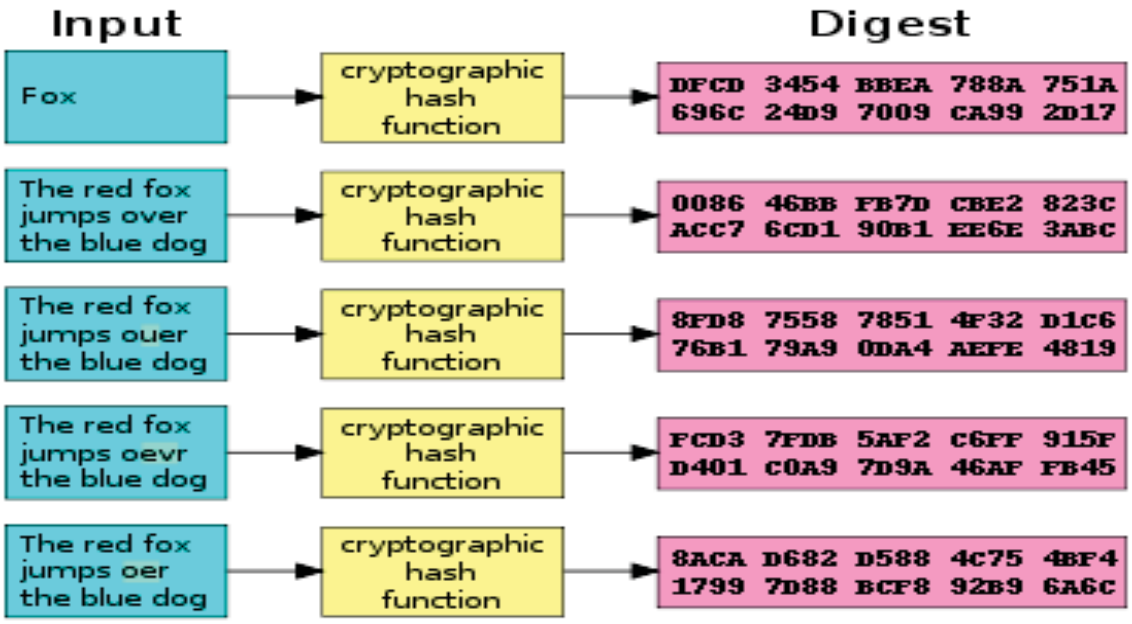


Figura 1-4 Representación función HASH (fuente: www.genbetadev.com)

1.2.1 Propiedades de las funciones HASH

Las funciones HASH deben ser capaz de resistir todos los ataques para vulnerar el sistema a través de ataques de fuerza bruta que en informática es el método que consta de utilizar la cantidad necesarias de combinaciones hasta dar con la correcta, y al mismo tiempo cumplir con las siguientes propiedades [2]:

- Unidireccionalidad: debe ser imposible encontrar la entrada del mensaje a través de su resumen.
- Comprensión: a partir de una longitud arbitraria en el mensaje en la entrada generar un mensaje de longitud fija a la salida.
- Facilidad de cálculos: se debe poder calcular de manera fácil la función HASH a partir del mensaje.
- Difusión: el resumen HASH debe ser un código complejo en cada uno de los bits del mensaje, si se llega a modificar solo un bit en la entrada, la función debe ser capaz de modificar al menos el 50% de los bits del resumen HASH.
- Colisión simple: debe ser computacionalmente imposible conocido el mensaje en la entrada de un HASH, encontrar otro mensaje que genere el mismo HASH.
- Colisión fuerte: será computacionalmente difícil encontrar dos mensajes distintos que genere la misma función HASH, debido a la resistencia fuerte a la colisión [2].

1.2.2 Usos de las funciones HASH

Esta función tiene una gran utilidad para la agilización en recuperación de los registros de datos (búsqueda simple en una sola dirección) que se utiliza para la validación de datos (sumas de comprobación) y para el cifrado de la información. Calculando el código HASH para los datos antes del almacenamiento o la transmisión y se vuelve a calcular al final para la comprobación de la integridad de estos datos, en el caso que no coincidan indica que los datos están dañados. Las funciones HASH se utilizan principalmente en la protección de datos [2].

Los principales casos de la utilización de HASH son:

- Detección de registros duplicados
- Localización de puntos cercanos entre si
- Verificar la integridad de los mensajes
- Verificación de contraseñas.
- Orden en el almacenamiento en los directorios.

2 Introducción al algoritmo MD

Esta presentación tiene como finalidad mostrar las características y evolución del código MD5, y cómo ha evolucionado en la protección de la vulnerabilidad de redes de computadores, y como se puede prevenir tomando una medida eficaz de encriptación a través de este esté algoritmo.

Este es el algoritmo más utilizado en este momento fue desarrollado por Ronald Rivest en 1995 y está basado en dos algoritmos anteriores MD2 y MD4 [3].

2.1 Algoritmo MD2

MD2 (acrónimo inglés de Message-Digest Algorithm 2, Algoritmo de Resumen del Mensaje 2) es una función de HASH criptográfica desarrollada por Ronald Rivest en 1989. El algoritmo está optimizado para computadoras de 8 bits. El valor hash de cualquier mensaje se forma haciendo que el mensaje sea múltiplo de la longitud de bloque en el ordenador (128 bits o 16 bytes) y añadiéndole un checksum (función para detectar accidentales en una secuencia de datos). Para el cálculo real, se utiliza un bloque auxiliar 48 bytes y una tabla de 256 bytes que contiene dígitos al azar del número pi(π). El mensaje se rellena si es necesario esta función de HASH criptográfica ya no se considera segura incluso a partir del año 2014, se mantiene en uso en infraestructura de claves públicas [3].

2.2 Algoritmo MD4

Es un algoritmo de resumen diseñado por el profesor Ronald Rivest del MIT. Implementa una función criptográfica de HASH para el uso en comprobación de integridad de mensajes. Producen un resume de 128 bits del mensaje [4].

El objetivo de diseño era la obtención de una función de resumen rápida, de simple diseño, compacta, optimizada para arquitecturas microprocesador (Intel en particular) y cuya seguridad fuera independiente de hipótesis no garantizadas (como la dificultad de factorización). Sin embargo, la sospecha de su posible vulnerabilidad ante ciertos ataques movió a su autor a modificar ligeramente su diseño para hacerlo más seguro, aunque ello supuso un algoritmo algo más lento. El resultado se conoce como MD5 [4].

2.2.1 Función

El algoritmo toma como entrada un mensaje de longitud arbitraria y produce como salida una" huella dactilar" o "resumen del mensaje" de 128 bits, se plantea que es computacionalmente inviable producir dos mensajes que produzcan el mismo resumen de mensaje. El MD4 está destinado a principalmente a aplicaciones de firma digital [4], donde el archivo grande debe ser comprimido de manera segura antes de ser encriptado con una clave privada, bajo el criptosistema de clave pública además está diseñado para ser bastante rápido en máquinas de 32 bits, y no requiere sustitución grande y puede codificarse de forma bastante compacta [4].

2.2.2 Colisiones

Es la posibilidad de que un mismo HASH sea asignado a más de un archivo, el MD4 quedó académicamente vulnerado en 1996 por el criptógrafo alemán Hans Dobbertin presentando algunas debilidades, pero no fue hasta el año 2004 que el chino Dengguo Feng afirmó que las colisiones de MD4 se pudieron calcular a mano, ésto no produjo sorpresa teniendo en cuenta las acusaciones de vulnerabilidad del sistema anteriormente [3].

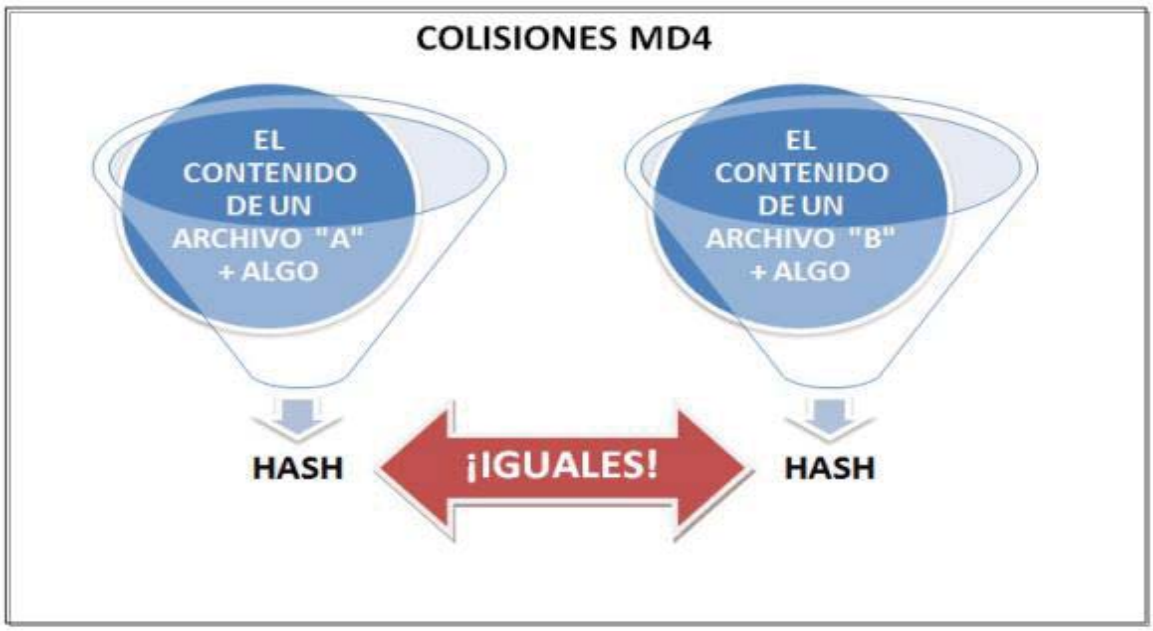


Figura 2-1 Representación colisión (fuente: wordpress.com)

2.2.3 Consecuencias de la vulnerabiliadad

La consecuencia más grande que se puede encontrar al tener en una red información distinta con el mismo HASH, es principalmente la confianza al momento de ser utilizado el algoritmo en la descarga de archivos que tienen alterado su contenido, ya que la función del HASH es la verificación de la información una vez descargada, y las pérdidas materiales que producen para las empresas que utilizan bajo este algoritmo archivos o información con fines lucrativos que por ningún motivo debe ser alterada [5].

3 Algoritmo MD5

Este algoritmo está diseñado por Ronald Rivest del MIT (Massachusetts Institute of Technology) en el año 1991, MD4 tuvo como pilares fundamentales en su diseño ser bastante rápido en las máquinas de 32 bits, además el algoritmo no necesita ninguna sustitución grande y puede codificarse de una forma bastante compacta [4]. Este algoritmo es una extensión del algoritmo MD4, donde se persigue el mismo objetivo siendo MD5 más lento, pero más conservador en su diseño. MD5 fue creado dado que MD4 estaba diseñado para operar de una forma más rápida de la necesaria, trabajando al límite de lo que se conoce como criptoanálisis exitoso, corriendo riesgos de ataque de fuerza, por otra parte, MD5 cede en velocidad, pero aumenta las probabilidades éxito en su seguridad. Después de recibir varias revisiones y sugerencia para su optimización MD5 se está abriendo al dominio público para una posible estandarización [4].

3.1 Funcionamiento

Cuando la seguridad del sistema operativo de un servidor parte fundamental del usuario, se debe tener estricto cuidado con las descargas que en él se introducen, pero muchas veces se instalan programas de la web que provienen de páginas que nos son oficiales, o por otro lado de páginas oficiales que llevan mucho tiempo en la web y ha sido alterada su información por usuarios malintencionado el cual puede haber añadido al archivo a instalar virus o troyanos [5].

Es importante la funcionalidad del algoritmo MD5 que se utiliza para varias aplicaciones, pero nos dirá con certeza en un caso como éste si la información que está no presenta ningún peligro para nuestro sistema operativo [4].

Explicando básicamente el funcionamiento del MD5, no tiene gran diferencia con el algoritmo MD4 ya que se persiguen los mismos objetivos generales, MD5 proporciona un código asociado a un texto o archivo, este código HASH o resumen del HASH viene unido a los archivos cuando se desean descargar [4].

Para poder ver el código MD5 se debe recurrir a programas especiales para poder analizar los archivos descargados, para poder obtener un código y así poder acudir a la página del instalador original del programa o archivo, obteniendo su código y comparando con el nuestro para así poder ver si la información que estamos descargando es fiable, y no ha sido alterada [4].

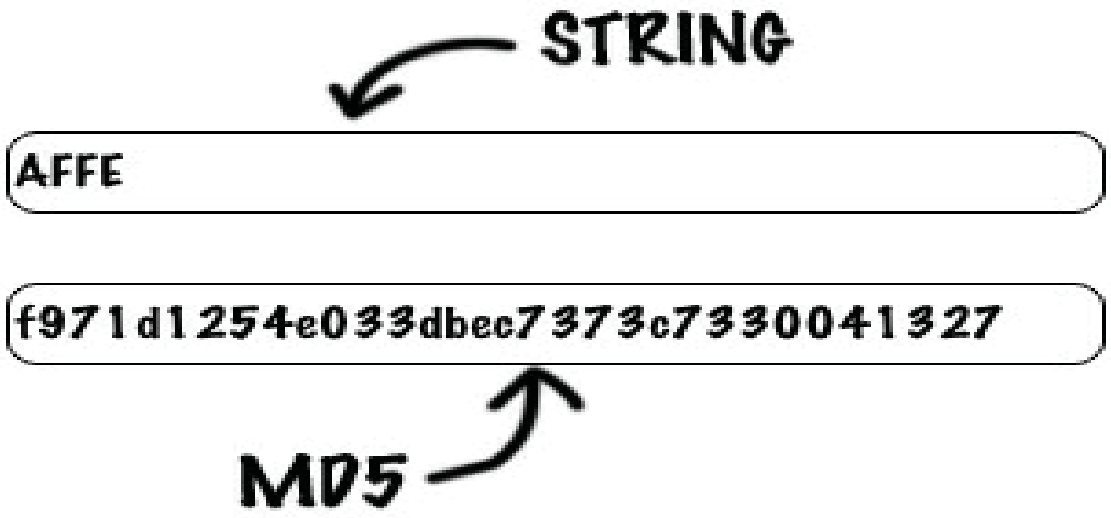


Figura 3-1Ejemplo de un generador MD5(fuente: http://www.antrax-labs.org)

En la figura 3-1 podemos apreciar cómo se genera un resumen del HASH a partir de un mensaje que tiene una entrada de 4 caracteres, a lo que obtenemos una salida de128 bits que como se explicó antes son fijos y queda explícito en la figura 3-2.

Figura 3-2Ejemplo de un generador MD5(fuente: www.taringa.net)

3.2 Aplicaciones

Aparte de verificar si la información es fiable tenemos otras aplicaciones importantes para el usuario que desee ocupar este algoritmo que le proporcionará seguridad, una aplicación muy interesante es que se pueden generar un algoritmo MD5 propio, para en el caso que el usuario quiera generar un código HASH para documentos propios lo puede hacer, así puede enviar este código para que otro usuario pueda verificar su integridad.

Otra aplicación que puede resultar interesante está en el firmware que tiene como eje principal además de proporcionarnos información sobre la seguridad del archivo, nos asegura que la instalación del archivo en cuestión este correcta y completo desde su descarga, esto es de gran utilidad cuando instalamos un firmware o un sistema operativo en nuestros dispositivos como puede ser en un router o actualizando una ROM, ya que realizar estos procesos con un archivo

dañado puede derivar en la pérdida total de los equipos o dispositivos ocupados por el usuario generando una gran pérdida de recursos y tiempo.

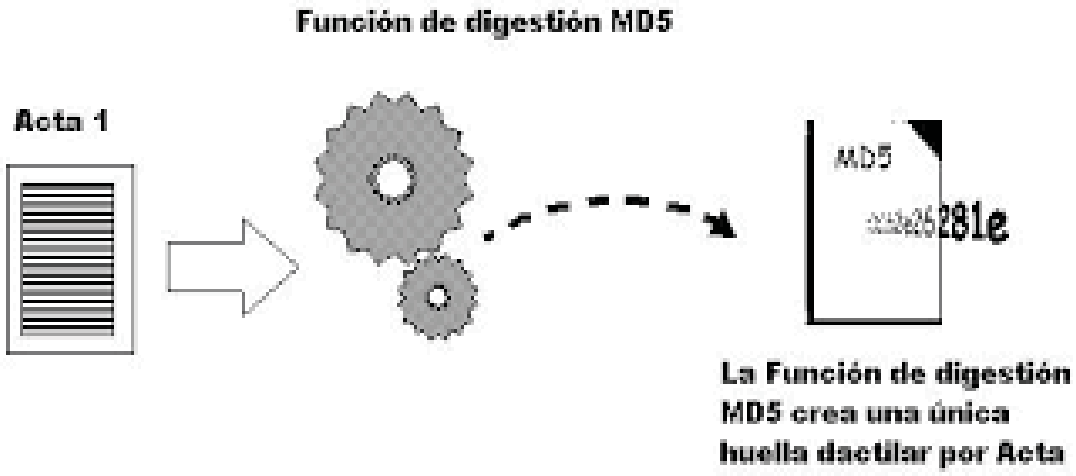


Figura 3-3 Aplicación algoritmo MD5(fuente: www.emaze.com)

MD5 permite comprobar la originalidad de un texto y que no haya estado sujeto a modificaciones, lo cual generaría que haya llegado a manos del receptor de forma alterada, con esto se pierde la confianza en la fuente o tener consecuencias graves a las personas que les compete esta información.

MD5 por otro lado está enfocado en la verificación de contraseñas y que correos electrónicos no hayan sido alterado utilizando claves públicas y privadas.

3.3 Generación del algoritmo

3.3.1 Codificación

La codificación MD5 consta de 128 bit que se representan con un número de 32 símbolos hexadecimales, los siguientes ejemplos corresponden a código ASCII tratado con MD5:

- MD5(“Generando un MD5 de un texto”) = 5df9f63916ebf8528697b629022993e8

Un pequeño cambio en el texto (cambiar ‘5’ por ‘S’) produce una salida completamente diferente.

- MD5(“Generando un MDS de un texto”) = e14a3ff5b5e67ede599cac94358e1028 Otro ejemplo sería la codificación de un campo vacío:
- MD5(“”) = d41d8cd98f00b204e9800998ecf8427e.

3.3.2 Algoritmo

El algoritmo MD5 posee 5 pasos claves para su generación los cuales son:

- Adición de bit

- Longitud del mensaje
- Inicializar el búfer MD
- Procesado del mensaje en bloques de 16 palabras
- Salida.

3.4 Seguridad

El algoritmo MD5 ha sido considerado criptográficamente uno de los más seguros, se basa en que es imposible tener dos archivos con un mismo código HASH, se ha demostrado en 2004 ciertas vulnerabilidades en el sistema, los investigadores chinos Xiaoyun Wang, Dengguo Feng, Xuejia Lai y Hongbo, descubrieron colisiones en el algoritmo, aparte de dar indicios de que 128 bits son pequeños para ser vulnerados con ataques de fuerza, el cual consiste en probar todas las combinaciones posible a hasta dar con la correcta.

4 Algoritmo SHA

SHA (Secure HASH Algorithm o algoritmo de HASH seguro) es una familia de funciones HASH de cifrado estandarizado por el Instituto Nacional de Normas y Tecnología (NIST) de EE.UU. La primera versión del algoritmo SHA se creó en 1993, y en la actualidad se conoce con el nombre de SHA-0, con lo que se evitan confusiones con las versiones posteriores. La segunda versión del sistema, publicada con el nombre de SHA-1, fue publicada dos años más tarde. Con el tiempo se han publicado más versiones de este algoritmo como SHA-1 en el 2001(formada por diversas funciones: SHA-224, SHA-256, SHA-384, SHA-512) y la más reciente SHA-3, que fue seleccionada en una competición de funciones HASH celebrada por el NIST en 2012 [6].

4.1 Algoritmo SHA-0

Es un resumen de mensaje de algoritmo denominado funciones HASH que toman como entrada un mensaje de longitud arbitraria y produce a la salida una “huella dactilar” de 160 bits o un resumen del mensaje de la entrada [6].

4.1.1 Seguridad

El Instituto Nacional de Estándares y Tecnología de los Estados Unidos (NIST) retiró SHA-0 en 1996, al no considerarlo seguro para cualquier transacción asociada con el uso de las operaciones criptográficas del gobierno de EE. UU, para la protección de información sensible no clasificada [6]. Hoy no se aconseja utilizar SHA-0 aunque continúan los análisis y se le ve como una versión más débil que SHA-1.

4.1.2 Resistencia a la colisión

El primer ataque a SHA-0 se produjo el año 1998 y enseñó que las colisiones se pueden encontrar en 2^61 operaciones realizando ataques de fuerza lo que significa que si se hace este número de pruebas se pueden encontrar colisiones vulnerando el sistema. El 2006, el NIST mostró un ataque mejorado capaz de encontrar colisiones en 2^36 operaciones [6].

En todos los casos, el resultado de las investigaciones conocidas demostró que SHA-0, no es resistente a colisiones. Como se esperaba, la fuerza de resistencia es significativamente menor a la de una función HASH ideal es decir 2^36 comparados con 2^80 [6].

4.1.3 Pre-imagen y segunda Pre-imagen

- Resistencia al cálculo de pre-imágenes: dado un valor de salida y = h(x), pero no el correspondiente valor x de entrada, debe ser prácticamente imposible encontrar el valor de x.
- Resistencia al cálculo de segundas pre-imágenes: dado un valor de salida y = h(x) y el correspondiente valor x de entrada, debe ser prácticamente imposible encontrar otra entrada $z \neq x$, tal que se genere la misma salida $h(z) = h(x)$ [6].

Los ataques de pre-imagen y segunda pre-imagen fueron publicados en formato reducido. Versiones de SHA-0(es decir menos de 80 rondas), dan a conocer que el margen de seguridad de SHA-0 es resistentes a estos ataques. En 2008 se mostró un ataque previo a la imagen en 49 de 80 rondas con complejidad de 2^159, y en 2009 un ataque previo a la imagen de en 52 de 80 rondas con complejidad de 2^156 lo que significa que soporta este número de pruebas antes de colisionar [7].

4.2 SHA-1

Es una función HASH criptográfica que toma una entrada de longitud arbitraria y genera una salida de valor 160bits (20bytes) conocido como resumen del HASH. Normalmente representa un número hexadecimal de 40 dígitos y fue diseñado por la Agencia Nacional de Seguridad de los Estados Unidos NSA y es un estándar Federal de procesamiento de información [8].

Este algoritmo se basa en los principios utilizados por el académico Ronald Rivest del MIT en los diseños de los algoritmos MD4 y MD5, pero tiene un diseño más conservador.

4.2.1 Desarrollo

SHA-1 fue desarrollado como parte del proyecto Capstone que es un proyecto a largo plazo del gobierno de los Estados Unidos para desarrollar estándares de criptografía para uso público y gubernamental. La especificación original del algoritmo fue publicada en 1993 y se tituló Secure HASH Standard por el Instituto Nacional de Estándares y Tecnología del gobierno norteamericano NITS. Esta versión es conocida como SHA-0 y en 1995 fue remplazada por la versión revisada SHA-1 que solamente difiere en la rotación de un único bit en el cronograma de mensaje de su función de compresión. La NSA (Agencia de Seguridad Nacional) argumentó que esto se hizo para corregir una falla en el algoritmo original que redujo su seguridad criptográfica, pero no proporcionó ninguna otra información.

4.2.2 Criptografía SHA-1

Se conforma por varias normas y protocolos ampliamente utilizados:

- **Transport Layer Security** (TLS) es un protocolo criptográfico que proporciona seguridad de comunicaciones a través de una red informática. Varias versiones de protocolos encuentran su uso en aplicaciones como navegación web, correo, mensajería instantánea y voz sobre IP [7]. Los sitios web pueden usar TLS para asegurar toda la información que

fluye entre sus servidores y navegadores web. Tiene que como objetivo principal proporcionar seguridad entre dos o más aplicaciones informáticas en comunicación [7].

- **Pretty Good Privacy** (PGP) es un programa de cifrado que se utiliza para proporcionar privacidad criptográfica y autentificación para la comunicación de datos PGP se utiliza para encriptar, firmar y descifrar textos, correos electrónicos, archivos, directorios y particiones de discos completos y para aumentar la seguridad de las comunicaciones por correo electrónico.
- **Secure Shell** (SSH) es un protocolo de red criptográfica para operar servidores de red de forma segura a través de una red no segura [7]. El ejemplo más conocido es la aplicación por la cual se efectúa el inicio de sesión remoto de los sistemas informáticos por parte de los usuarios. SSH proporciona seguridad en una arquitectura cliente-servidor, conectando una aplicación SHH con un servidor SHH. Las aplicaciones comunes incluyen el inicio de sesión remoto de la línea de comandos y la ejecución remota de comandos, pero cualquier servicio de red se puede asegurar con SHH [7].
- **S/MIME** es un estándar para el cifrado de clave pública y la firma de datos MIME, y proporciona los servicios de seguridad criptográfica para aplicaciones de mensajería electrónica como la autentificación, integridad del mensaje, no repudio de origen, intimidad y seguridad de datos [7].
- **Internet Protocolo Security** (IPsec) es un conjunto de protocolo de red seguros IPv4 que autentica y encripta paquetes de datos a través de una red IPv4 [7]. Ipsec incluye protocolos para establecer la autentificación mutua entre al comienzo de la sesión de encriptación de claves criptográficas utilizadas durante la sesión; se pueden proteger los flujos de datos entre un par hosts (host-a-host); entre par de puertas de enlace de seguridad (red-a-red) o entre la puerta de enlace de seguridad y un host (red-a-host) [7]. IPsec usa servicios de seguridad criptográfica para proteger las comunicaciones a través de las redes de protocolo de internet, además admite la autenticación entre pares de a nivel de red, de origen de datos, integridad de los datos, confidencialidad (cifrado) y la protección de reproducción [7].

Estas aplicaciones también usan tanto en MD5, como SHA-1 y descienden directamente de MD4. El algoritmo SHA-1 también ha sido utilizado en consolas de Nintendo Wii para las verificaciones de contraseñas y firmas durante el arranque. Una de las principales motivaciones para la publicación del algoritmo SHA-1 fue el estándar de firma digital, en el cual está incorporado [7].

4.2.3 Integridad de datos

Una función de los algoritmos SHA permite también no solo garantizar la seguridad sino la integridad de la información. Los sistemas de control de revisión como Git, Mercurial y Monotone ocupan SHA-1, no por temas de seguridad, sino para evitar mediante revisiones y garantías sobre los datos, alteraciones debido a daños accidentales [8].

4.2.4 Consideraciones de seguridad SHA-1

El NIST ha recomendado no utilizar este algoritmo para firmas digitales después del 31 de diciembre de 2010, y especificó que no se debe utilizar para firmas digitales de agencias del

gobierno de EE. UU. Señaló que también no se utilizará para “protección de personas sensibles” y archivos no clasificados después del 31 de diciembre del 2013.

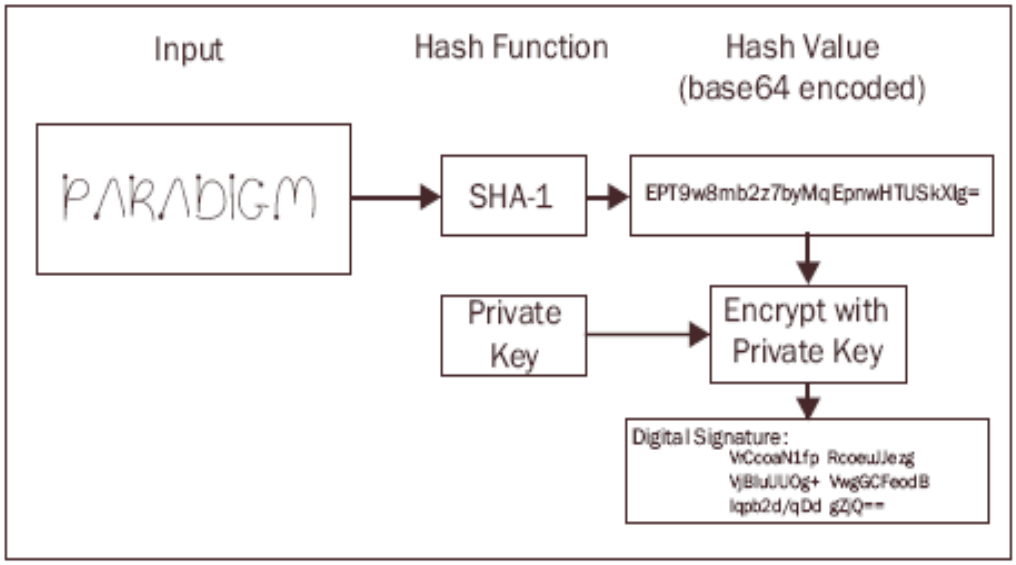


Figura 4 1 Representación algoritmo SHA-1(isla1md.wordpress.com)

4.2.5 Resistencia a la colisión

El primer ataque a SHA-1 fue publicado en 2005. Se describió como un ataque teórico a una nueva versión SHA-1 que fue reducido a 53 rondas. En el siguiente mes se mostró colisiones en las 80 rondas completas en 2 ^ 69 operaciones. [8]Conocido esto, se desarrollaron nuevos métodos de análisis para mejorar el ataque presentado en 2005. Pero no se han presentado resultados de mejoras encontrados. En cualquier caso, los resultados de las investigaciones conocidos indican que SHA-1 no es tan resistente a colisiones como se esperaba, la fuerza de seguridad es mucho menor a la de una función hash ideal es decir 2 ^ 69 en comparación en 2 ^ 80 [8].

4.2.6 Pre-imagen y segunda resistencia a la Pre-imagen

No se conocen los ataques previos a la imagen o a la segunda pre-imagen que son específicos para el algoritmo SHA-1 de ronda completa [8]. Encontrar una segunda pre-imagen requiere menos de 2 ^ n cálculos con n = 160, en el caso de SHA-1 lo hará tomando 2 ^106 cálculos para encontrar una segunda pre-imagen en un mensaje de 60 bytes [8].

Los criptógrafos, en ausencia de ataques completos, consideran ataques reducidos para obtener pistas sobre la resistencia de un algoritmo. Estos ataques de rondas reducidas, donde el número de rondas es una poco menos que las rondas completas, no han demostrado que tengan relación una con otra, pero el ataque de ronda reducida indica un cierto margen de seguridad [8]. Por ejemplo, si el ataque conocido está en la ronda 60 de 80 eso indica que existen 20 rondas para resistir a ataques mejorados. Sin embargo, el número de rondas que puede alcanzar un ataque y el número de rondas definidas para el algoritmo tienen una relación no lineal, en otras palabras,

no proporcionan una prueba matemática, ya que los ataques de red reducida indican que tan fuerte es un algoritmo frente a cierto ataque [8].

Los ataques de imagen y pre-imagen publicado en formato reducido menos de 80 rondas indican que SHA-1 conserva un margen de seguridad significativo frente a estos ataques, en 2009 el NIST mostro un ataque de pre-imagen en la 48 de las 80 rondas con complejidad de 2 ^159 [8].

4.2.7 Pseudocódigo SHA-1

El algoritmo consta de los siguientes pasos para su ejecución:

1. Inicialización de variables:
    - h0 = 0x67452301
    - h1 = 0xEFCDAB89
    - h2 = 0x98BADCFE
    - h3 = 0x10325476
    - h4 = 0xC3D2E1F0.
    - ml = longitud del mensaje en bits (siempre un múltiplo de la cantidad de bits en un carácter).
2. Procesamiento:
    - añada el bit '1' al mensaje, por ejemplo, agregando 80 ceros si la longitud del mensaje es un múltiplo de 8 bits.
    - añada $0 \leq k < 512$ bits '0', de modo que la longitud del mensaje resultante en bits es congruente con $64 \equiv 448 \pmod{512}$.
    - añada ml, la longitud del mensaje original, como un entero de 64 bits big-endian(formato que ordena los bytes del más al menos significativo). Por lo tanto, la longitud total es un múltiplo de 512 bits.
3. Procesamiento del mensaje en fragmentos sucesivos de 512 bits:
    - Romper mensaje en trozos de 512 bits
    - Por cada trozo dividir el fragmento en dieciséis palabras de 32 bits big-endian w [i], $0 \leq i \leq 15$.
    - Se amplían las 16 palabras de 32 bits en ochentas palabras de 32 bits, w [i] = (w [i-3] xor w [i-8] xor w [i-14] xor w [i-16]).
4. Inicializar el valor hash para este fragmento:
    - a = h0
    - b = h1
    - c = h2
    - d = h3
    - e = h4
5. bucle principal:

❖ sí $0 \leq i \leq 19$ entonces

- f = (b y c) o ((no b) yd)
- • k = 0x5A827999.

Else si $20 \leq i \leq 39$

- f = b xor c xor d
- k = 0x6ED9EBA1

Else si 40 ≤ i ≤ 59

- f = (b y c) o (b y d) o (c y d)
- k = 0x8F1BBCDC

Else si 60 ≤ i ≤ 7

- f = b xor c xor d
- k = 0xCA62C1D6

temp = (a leftrotate 5) + f + e + k + w [i]

- e = d
- d = c
- c = b leftrotate 30
- b = a
- a = temperatura

6. Se agrega el hash a este fragmento hasta el resultado:
   - h0 = h0 + a
   - h1 = h1 + b
   - h2 = h2 + c
   - h3 = h3 + d
   - h4 = h4 + e
     ❖
7. Produzca el valor de HASH final (big-endian) como un número de 160 bits:
   - hh = (h0 leftshift 128) o (h1 leftshift 96) o (h2 leftshift 64) o (h3 leftshift 32) o h4.

El número hh es el resumen del mensaje, que se puede escribir en hexadecimal base 16 pero a menudo se escribe utilizando la codificación de texto binario a ASCII de Base 64 (base64 son una cadena de caracteres que solo contienen caracteres az, AZ, 0-9, + y / y se utilizan a menudo en situaciones en las que se envía información no textual a través de un protocolo de transmisión de solo texto.).

Los valores constantes usados son elegidos para no ser nada en mis números de manga: las constantes de cuatro vueltas k son 2x30 veces las raíces cuadradas de 2, 3, 5 y 10. Los primeros cuatro valores iniciales para h0 a h3 son los mismos con MD5 algoritmo, y el quinto (para h4) es similar.

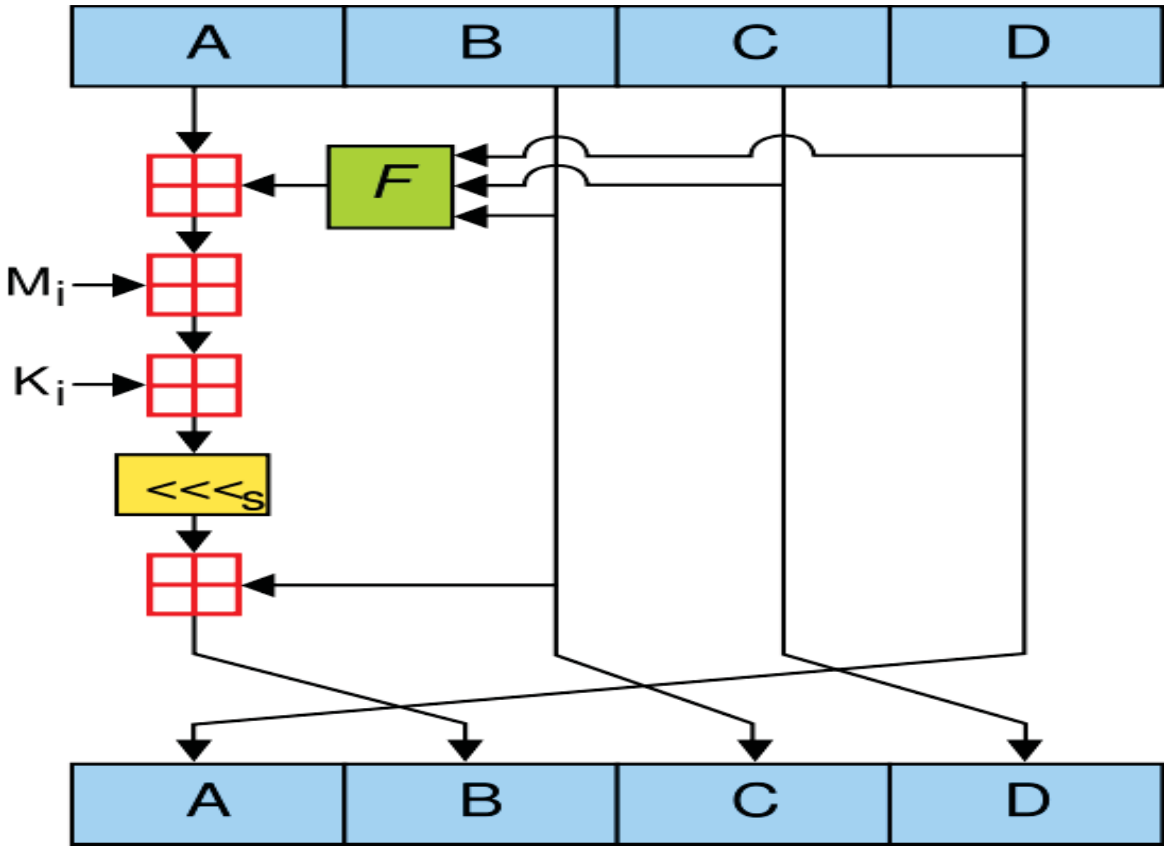


Figura 4-1 Diagrama del algoritmo SHA-1(en.wikipedia.org)

4.3 SHA-2

4.3.1 ¿Que es SHA-2?

Es un conjunto de funciones HASH criptográficas (SHA-224, SHA-256, SHA-384, SHA-512), diseñadas por la Agencia de Seguridad Nacional (NSA) y publicada en 2001 por el Instituto Nacional de Estándares y Tecnología (NIST) como un Estándar Federal de Procesamiento de la Información (FIPS) [9].

La función HASH más utilizada habitualmente es SHA-256. Generalizando se puede decir que SHA-2 es equivalente a SHA-256 [9].

4.3.2 Seguridad

La seguridad proporcionada por un algoritmo HASH depende totalmente de la capacidad de generar un único valor para un conjunto de datos dados [9]. En 2005 se identificaron fallas en el algoritmo SHA-1, permitiendo que se evidenciara una debilidad matemática y una necesidad de elaborar una función HASH más fuerte. Si bien SHA-2 opera de una forma parecida a SHA-1 no se han presentado a la fecha vulneraciones y ataques satisfactorios a SHA-2. La competición de funciones hash de la NIST seleccionó una nueva función hash en 2012 bajo el nombre de SHA-3 a diferencia de las versiones anteriores de SHA esta última no deriva de este algoritmo [9].

4.3.3 SHA-256

Es un algoritmo de encriptación HASH de 64 dígitos hexadecimal, casi único de un tamaño fijo de 256 bits (32 bytes). Este HASH se calcula solo en una dirección y no se puede decodificar de vuelta. No es mucho más complejo de codificar que SHA-1, y aun bajo esta característica no se ha visto amenazado de ninguna manera. La clave, al poseer 256 bits, se convierte en una muy buena función socio para AES (Algoritmo de Cifrado Simétrico). El Instituto Nacional de Estándares Tecnológicos NIST proporciona una serie de vectores de pruebas para verificar la correcta implantación.

4.3.4 Aplicaciones de SHA-256

Este algoritmo se utiliza en un gran número de herramientas de seguridad y protocolos algunos de ellos son:

- TLS
- SSL
- PGP
- SSH
- S/MIME
- Ipsec
- Bitcoin.

Donde en el protocolo Bitcoin SHA-256 es utilizado para la generación de claves o direcciones públicas y también en la minería Bitcoin lo cual es una moneda criptográfica que necesita seguridad informática.

4.3.5 Validación oficial

Para una función en la cual L es el número de bits del mensaje tratado, encontrar un mensaje que corresponde con el mensaje tratado siempre se puede conseguir empleando una búsqueda de fuerza bruta con 2L evaluaciones. A esto se le denomina ataque pre-imagen y puede ser práctico o no dependiendo del valor de L y del entorno de computación empleado. El segundo criterio, encontrar dos mensajes diferentes que producen el mismo mensaje tratado, se le conoce como colisión, requiere únicamente una media de 2L/2 evaluaciones si se emplea ataque malicioso [7].

Las implementaciones de todas las funciones de seguridad aprobadas por FIPS son oficialmente validadas mediante un programa con categoría CMVP (Certified Measurement & Verification Professional) y necesariamente ejecutadas por la Instituto Nacional de Estándares y Tecnología (NIST) y la Communications Security Establishment (CSE). Para una verificación informal, fue dispuesto en la página web del NIST un paquete para generar un alto número de vectores de prueba; sin embargo, los resultados de verificación no pueden remplazar de ninguna forma a la validación formal de CMVP, el cual es requerido por ley para ciertas aplicaciones [7].

4.3.6 Pseudocódigo SHA-256

Para el caso de este algoritmo ocurre un gran incremento en el número de bits que se puede apreciar a la hora de mezclar los bits de las palabras.

1) Inicialización de los valores HASH
2) Inicialización del array con las constantes de las rondas
3) Preprocesamiento
4) Procesamiento del mensaje en sucesivos trozos de 512 bits
5) Expansión de las primeras 16 palabras hasta completar las 48 palabras w [16…63] del array del mensaje
6) Inicialización de las variables de trabajo con los valores HASH actuales
7) Inserción del trozo comprimido al valor HASH actual
8) Producción del valor final del HASH.

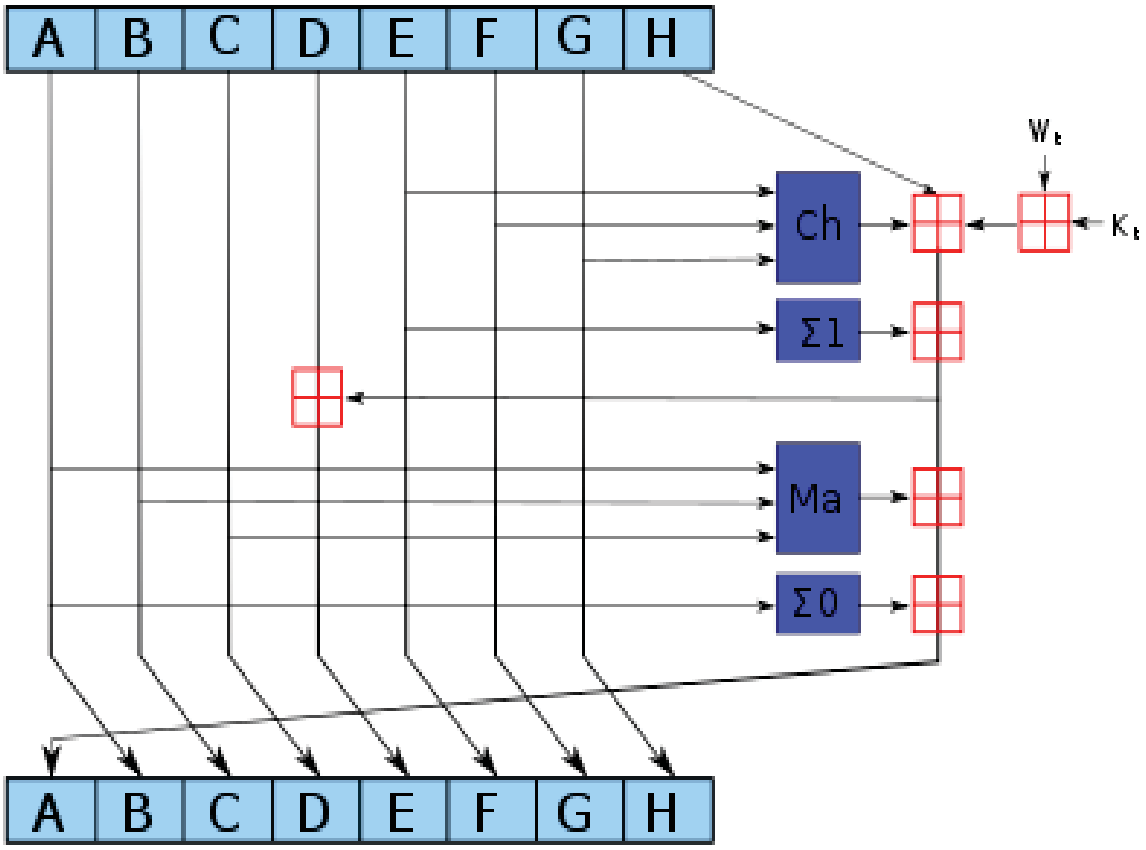


Figura 4-2 Diagrama algoritmo SHA-2(es.wikipedia.org)

4.3.7 SHA-1 v/s SHA-2

Algorithm	Output size (bits)	Internal state size (bits)	Block size (bits)	Max message size (bits)	Rounds
SHA1	160	160 (5 × 32)	512	$2^{64} - 1$	80
SHA256	256	256 (8 × 32)	512	$2^{64} - 1$	64

Tabla 4-1 Propiedades de algoritmos(www.keycdn.com)

En la figura 1-2 se muestra el largo del resumen del HASH SHA-1 respecto al SHA-256 este último de longitud más amplia que el primero, haciéndolo muy seguro y no vulnerable.

- **SHA1** – da39a3ee5e6b4b0d3255bfef95601890afd80709
- **SHA256** – e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855

Figura 4-3 Ejemplo comparativo SHA (www.keycdn.com)

5 Implementación del algoritmo MD5

5.1 Pseudocódigo

Se empieza suponiendo que tenemos un mensaje de “b “bits en la entrada, y que nos gustaría encontrar su resumen de HASH, donde “b” ‘es un valor arbitrario o negativo, pero puede ser cero.

$$m0\ m1\ ...\ m\{b-1\}$$

Y se efectúan los siguientes cinco pasos para obtener el resumen del HASH [10].

5.1.1 Adición de bits

El mensaje será extendido hasta que su longitud sea congruente con 448 módulo 512. Esto significa que si se le resta 448 a la longitud del mensaje tras este paso se obtiene un múltiplo de 512, este paso se realiza siempre incluso cuando el mensaje ya es congruente con 448, módulo 512 [10].

La extensión se realiza añadiendo un solo bits “1” al mensaje, y después añadiendo bits “0” hasta que el mensaje sea congruente con 448, módulo 512. En todos los mensajes se añaden al menos un bit y como máximo 512 [10].

5.1.2 Longitud del mensaje

Un entero de 64 bits que represente la longitud de “b” del mensaje (antes de añadir los bits de relleno) se concatena al resultado del paso anterior. En el caso que b sea menor a 2^{64}, entonces solo los bits de menor pesos de “b” serán utilizado.

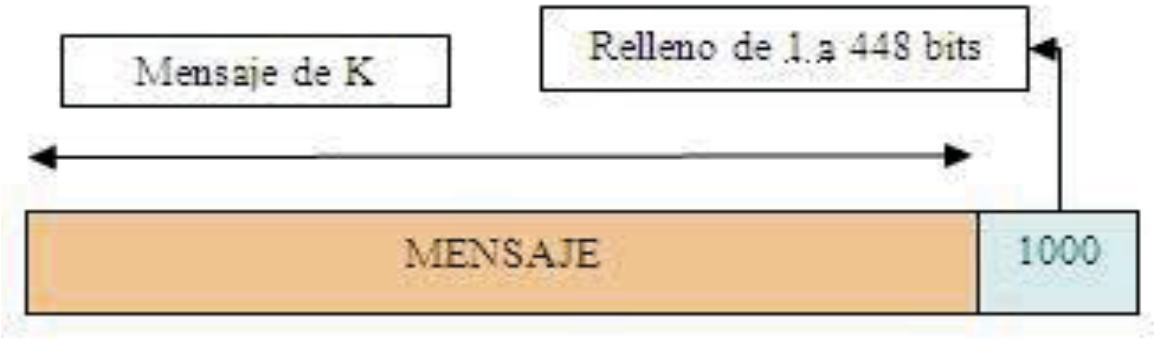


Figura 5-1 Representación longitud del mensaje(revistatelematica.cujae.edu.cu)

En este punto el mensaje que resulta después de rellenar con los bits y con “b” se obtiene una longitud que es múltiplo exacto de 512bits. A su vez, la longitud es múltiplo de 16 palabras cada una de 32 bits, se denotará las palabras del mensaje resultante M [0... N-1], donde N es múltiplo de 16.

5.1.3 Inicializar el búfer MD

Un búfer de cuatro palabras (A, B, C, D) se usa para calcular el resumen del mensaje. Aquí cada una de las letras A, B, C, D representa un registro de 32 bits. Estos registros se inicializan con los siguientes valores hexadecimales, los bytes de menor peso primero [11].

- palabra A: 01 23 45 67
- palabra B: 89 ab cd ef
- palabra C: fe dc ba 98
- palabra D:76 54 32 10

5.1.4 Procesado del mensaje en bloques de 16 palabras

En el algoritmo MD5 opera en un estado de 128 bits divido en palabras de 32 bits, estas están denotadas como A, B, C, D. estos son inicializados con ciertas constantes fijas, y el algoritmo principal utiliza de 512 bits para modificar el estado [4]. El procesamiento de un mensaje cuenta con 4 etapas muy similares, esta se denominan rondas, cada una de las rondas se componen de 16 operación similares basadas en una función no lineal, principalmente adición modular y rotación de bits.

Las rondas poseen cuatro operaciones posibles y se utiliza una para cada ronda.

$$F(X,Y,Z) = (X \wedge Y) \vee (\vec{X} \wedge Z)$$

$$G(X,Y,Z) = (X \wedge Z) \vee (Y \wedge \bar{Z})$$

$$H(X,Y,Z) = X \oplus Y \ \oplus Z$$

$$I(X,Y,Z) = \ Y \oplus (X \vee \bar{Z})$$

Se guardan los valores de las rondas A como AA, B como BB y C como CC, D como DD.

Ronda 1.
/* [abcd k s i] denotarán la operación

a = b + ((a + F (b, c, d) + X[k] + T[i]) <<< s). */

/* Hacer las siguientes 16 operaciones. */
[ABCD 0 7 1] [DABC 1 12 2] [CDAB 2 17 3] [BCDA 3 22 4]
[ABCD 4 7 5] [DABC 5 12 6] [CDAB 6 17 7] [BCDA 7 22 8]
[ABCD 8 7 9] [DABC 9 12 10] [CDAB 10 17 11] [BCDA 11 22 12]

[ABCD 12 7 13] [DABC 13 12 14] [CDAB 14 17 15] [BCDA 15 22 16]

Después se realizan las siguientes para el incremento de cada uno de los registros por el valor que tenían antes de inicializar este bloque.

- A = A+ AA
- B = B+ BB
- C = C+ CC
- D= D+DD.

5.1.5 Salida

El resumen del HASH es producido por A, B, C y D se forma comenzando por el byte de menor peso de A y se acaba con el byte de mayor peso de D, y se obtiene una salida de 128 bits.

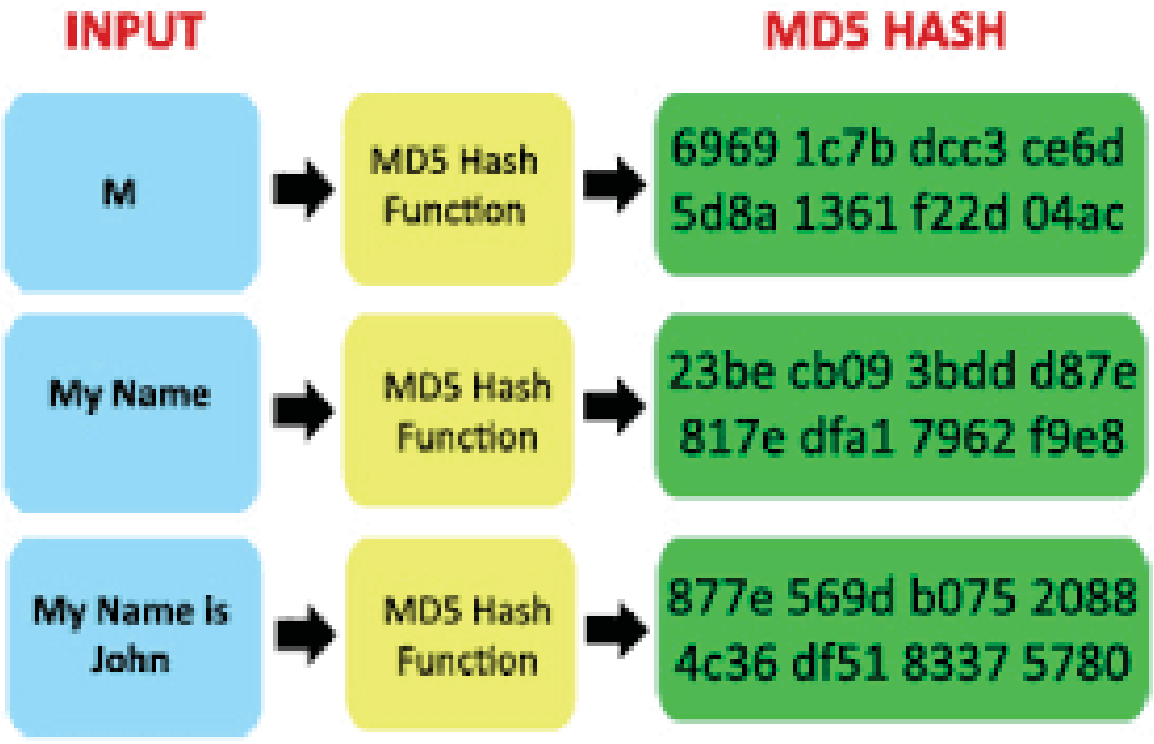


Figura 5-2 Representación salida MD5 (www.emaze.com

5.2 Lenguajes a usar

Se usará el lenguaje Virtual Pascal y Assembler.

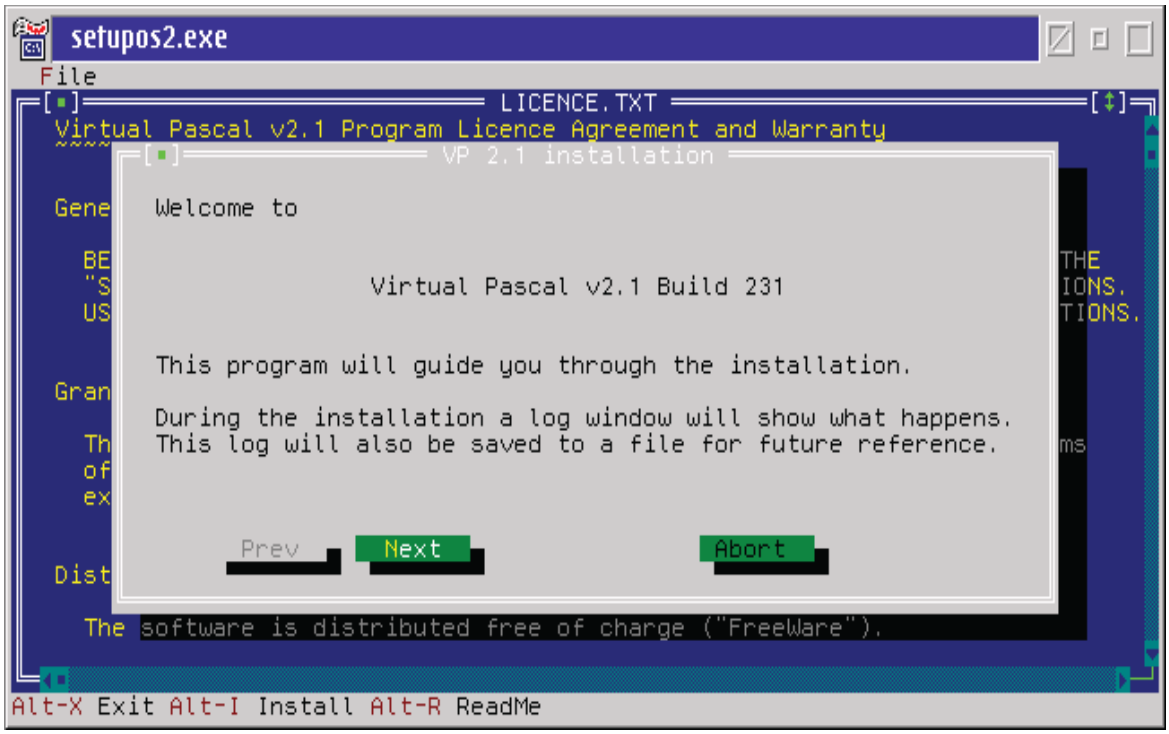


Figura 5-3 Programador Virtual Pascal(www.unocero.com)

Características**:**

- Sistema operativo: MS-DOS, CP/M, CP/M-86, Microsoft Windows
- Desarrollador: Borland
- Estado actual: descontinuado
- Plataforma: 8080 desarrollo integrado
- Lanzamiento inicial: 1983 y 1992
- Licencia: LC
- Autor: Borland.

6 Barra del menú principal

6.1 Secuencias de teclas (hotkeys)

En esta sección se ilustrará cada una de las opciones que nos da el compilador de virtual Pascal para generar diversas operaciones dentro del programa, éste puede ser desde guardar el archivo hasta pedir ayuda sobre algún comando desconocido. Con la tecla F10 nos situamos en el menú principal desde donde podemos pulsar cualquiera de las opciones de la barra [11].

Las teclas de función y secuencias “Shift”, “Alt” y “Ctrl”, junto con otras teclas ejecutan unas ciertas funciones. Por ejemplo,

- ALT+ primera letra de cualquier orden del menú principal
- ALT-C activa orden Compile
- ALT-E activa el editor: orden Edit [11].

6.2 File

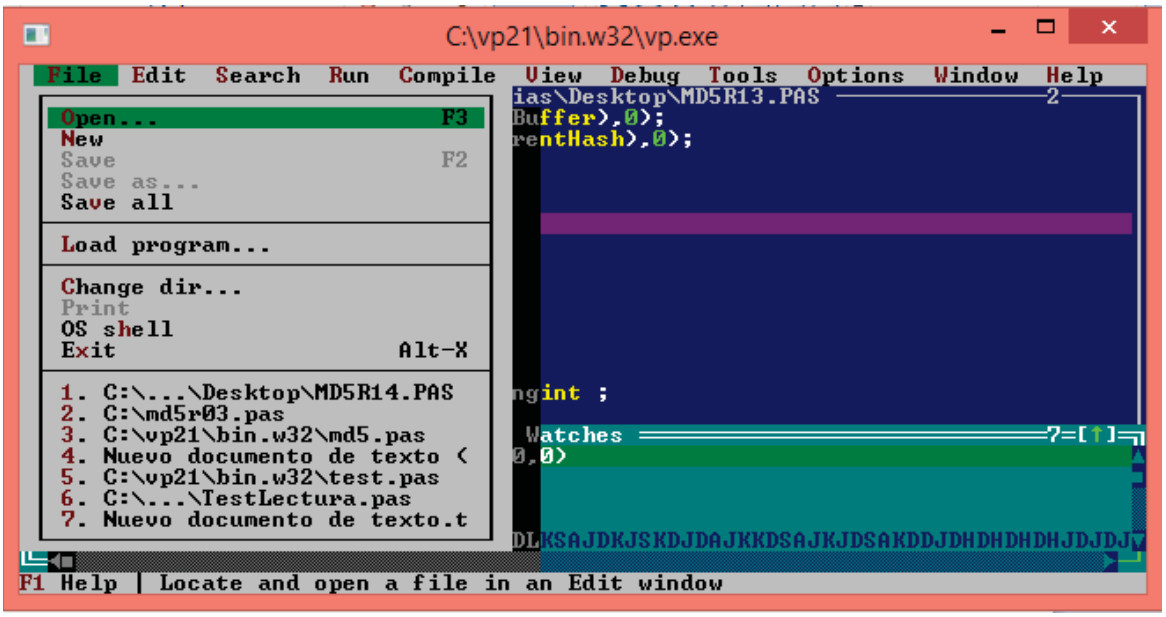
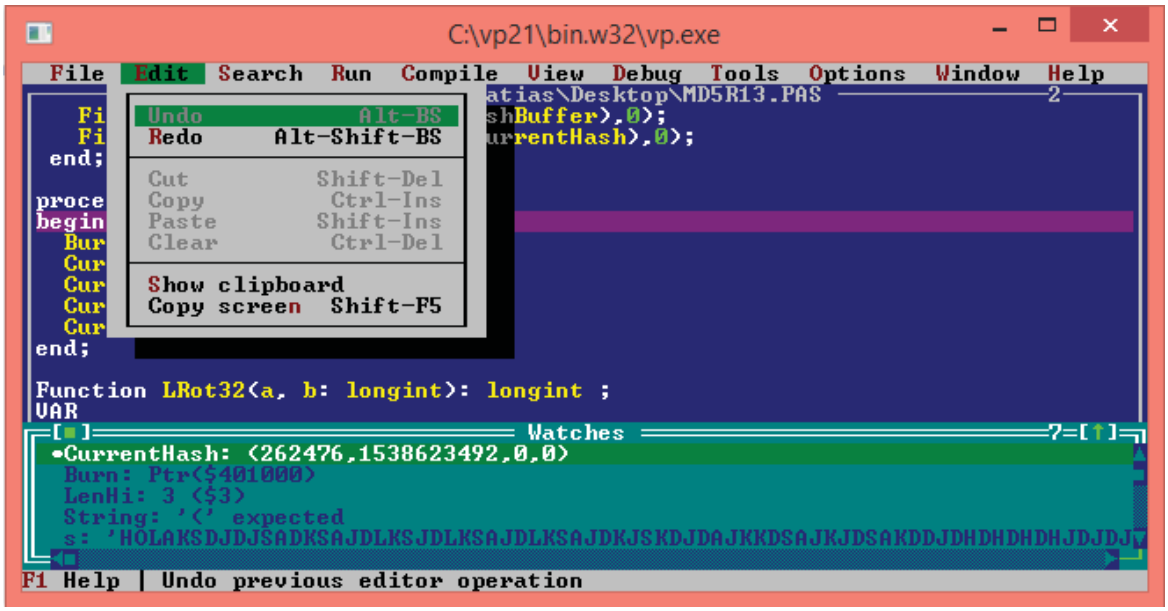


Ilustración 1 Opción “File”

6.3 Edit



6.4 Search

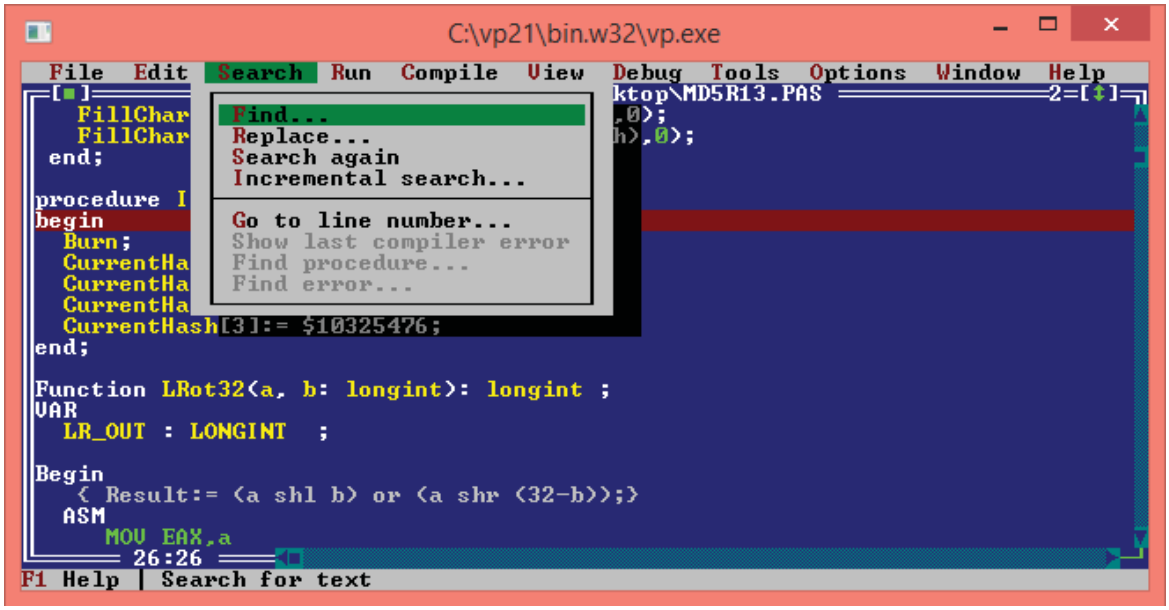


Ilustración 2 Opción “Search”

Este menú nos da las siguientes opciones de comandos que para poder actuar sobre el programa de una forma más expedita y rápida.

- **Find** es una herramienta muy útil que nos permite la búsqueda de ficheros dentro de una estructura de directorios con una gran cantidad de opciones para la búsqueda. como se muestra en la ilustración [12].

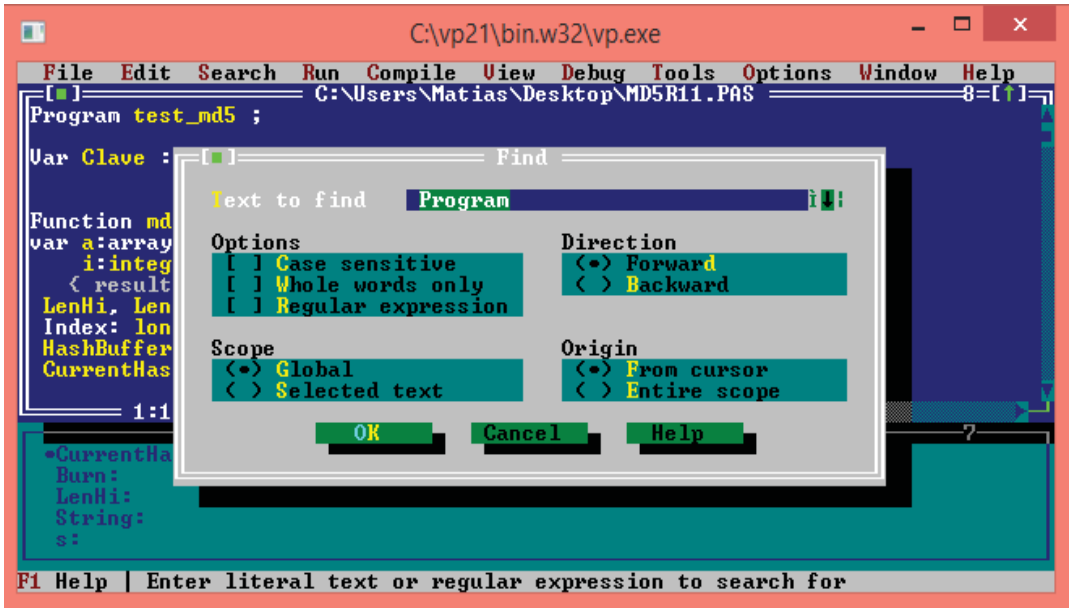


Ilustración 3 Opción “Find”

- **Replace** devuelve una copia de la cadena en la que las ocurrencias antiguas se han reemplazado por unas nuevas. Y nos da la opción “change all” la cual nos permite modificar todo como se muestra en la ilustración [12].

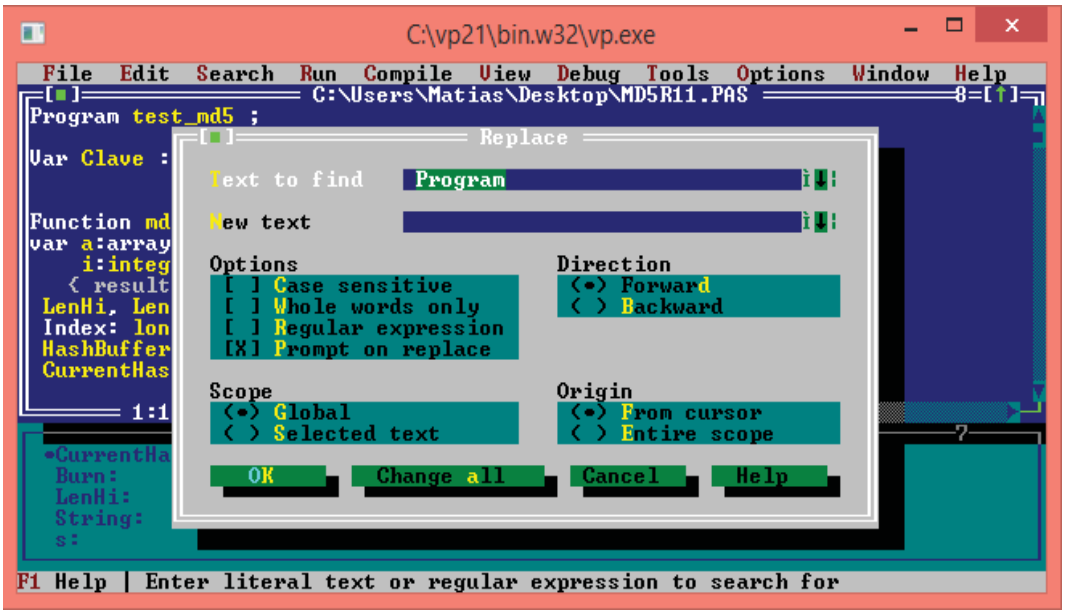


Ilustración 4 Opción”remplace”

- **Go to line number** este comando nos permite ir a una línea específica del programa ingresando su número lo cual hace más expedita la búsqueda ya sea para modificar, eliminar parte del programa. [12]

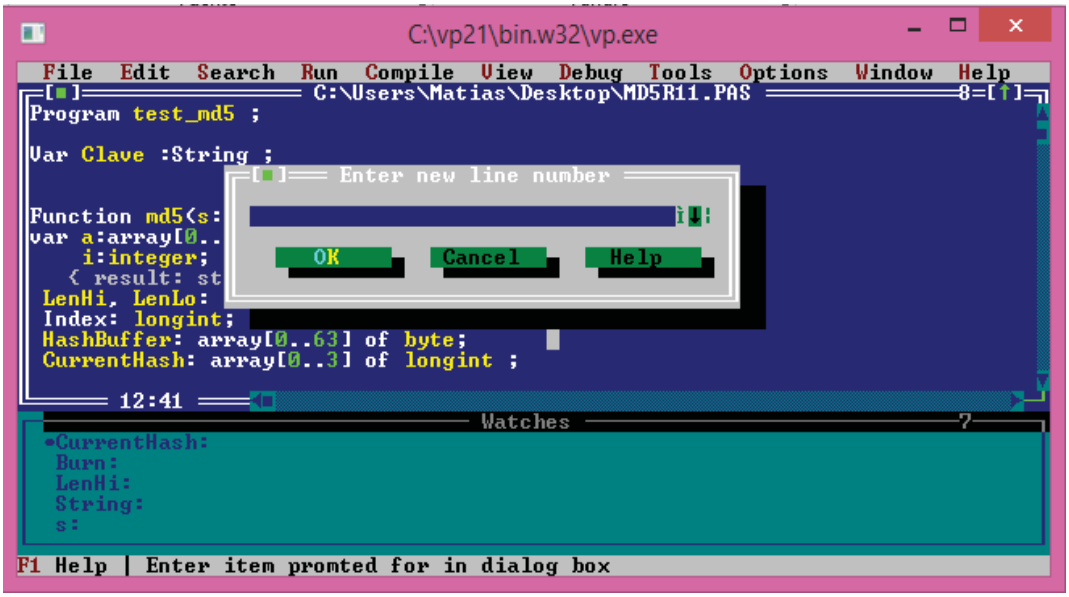


Ilustración 5 Opción “Go to line number”

6.5 Run

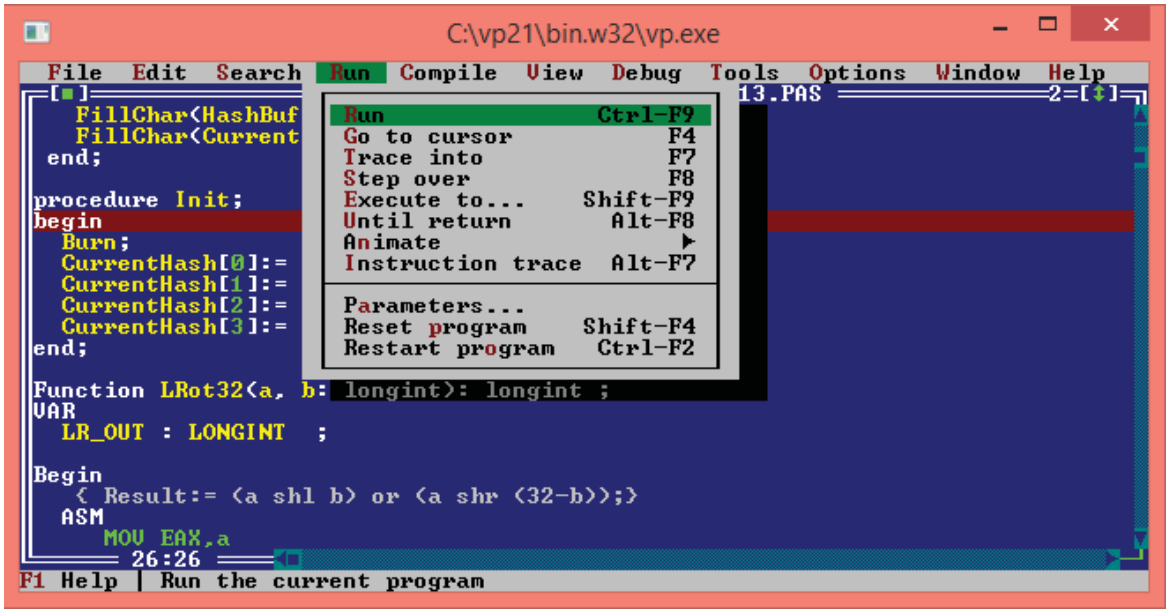


Ilustración 6 Opción “Run”

En el menú Run encontramos una de las partes más importantes, la cual es la que nos permite arrancar el programa y hacer pruebas de este verificando errores en su contenido, los comandos

de este menú se activan de forma rápida desde el teclado y no es necesario ingresar al menú [12].

- **Run** como se muestra en la ilustración 7 se activa presionando las teclas Ctrl-F9 la cual arranca el programa mostrando si posee algún error, en el caso que no se encuentre ninguno lo ejecuta sin problema, como se muestra en la siguiente ilustración [12].

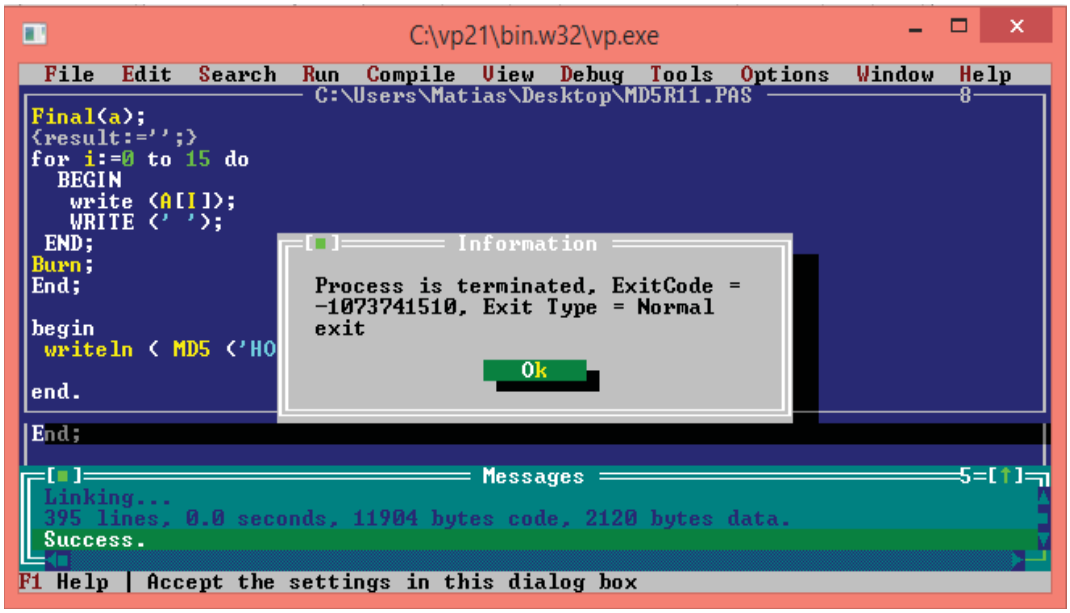


Ilustración 7 Opción "Run"

- **Go to cursor** este comando nos permite ejecutar por pantalla los resultados de cada una de las acciones del programa, éstas pueden ser de cualquier parte o condición del programa, como se mostrarán en las siguientes ilustraciones donde se ejecutará la última condición del programa que contiene el resultado de la semilla arbitraria introducida para ser encriptada [12].

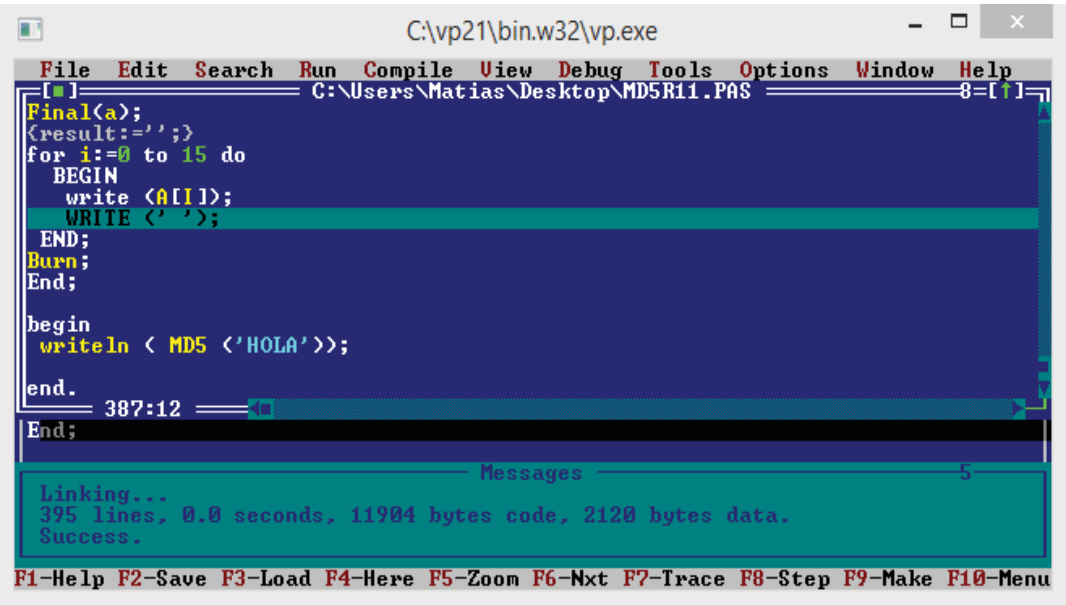


Ilustración 8 Matriz del Hash

Presionando F4 se ejecutará cada uno de los resultados dentro de la matriz que contiene 16 espacio que contienen el resumen del HASH de 128 bits, el cual cada uno de los caracteres posee

8 bits [12].

Ilustración 9 “Resultado primer espacio”

Ilustración 10 “segundo espacio de la matriz”

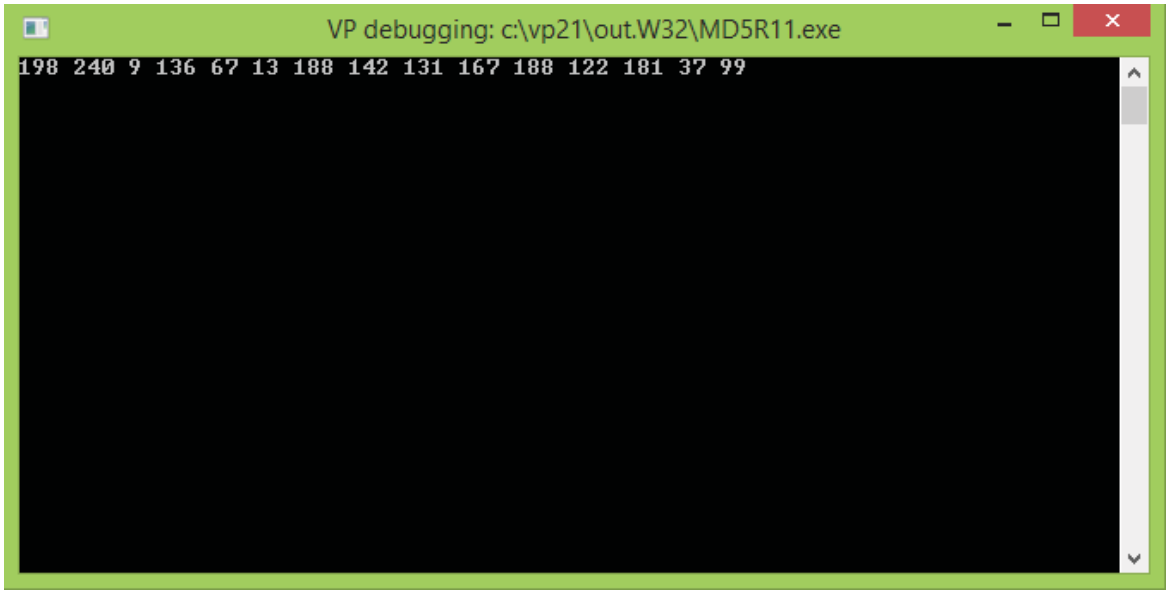


Ilustración 11 “Resultado 15 de la matriz”

En la ilustración 12 se puede apreciar los resultados de la matriz ejecutados uno por uno con el comando F4, el cual permite en una eventual revisión saber si alguno de los resultados no corresponde a la semilla ingresada [12].

- **Trace into** este comando se activa presionando F7 se puede ir línea a línea por el programa entrando a funciones y procedimientos que estos contienen, lo cual permite

revisar de una forma general el programa o de manera específica cierta parte del programa [12].

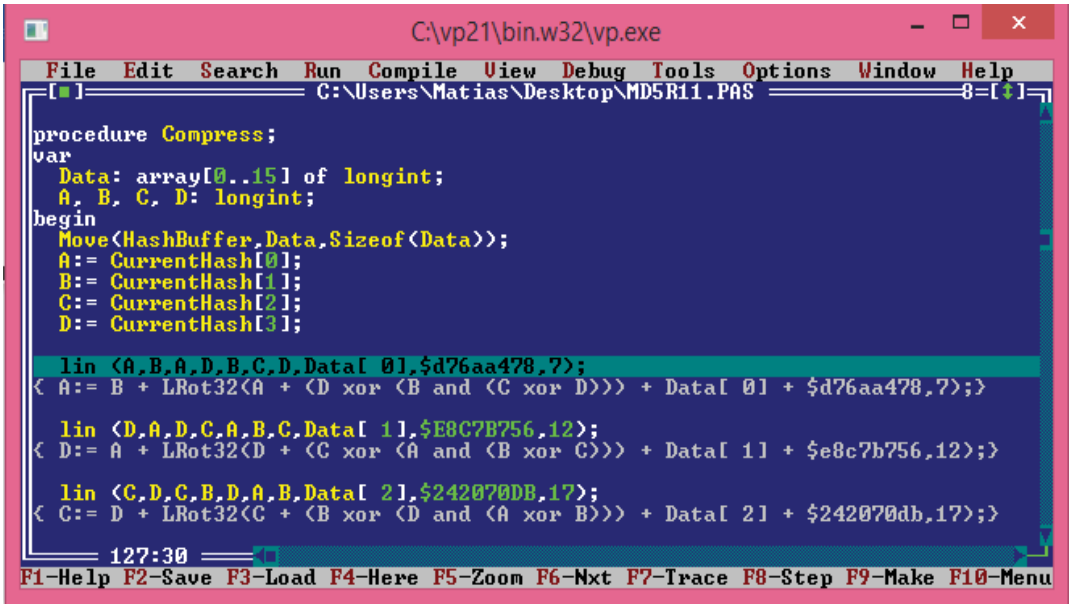


Ilustración 12 línea a línea “compress”

En la ilustración 13 se muestra cómo se ingresa línea a línea para revisar cada una de las rondas donde se van rellenando las matrices y mezclando el código generando el resumen del HASH del algoritmo MD5, lo cual al presionar F7 debería el cursor saltar a la siguiente línea como se muestra en la ilustración 14 [12].

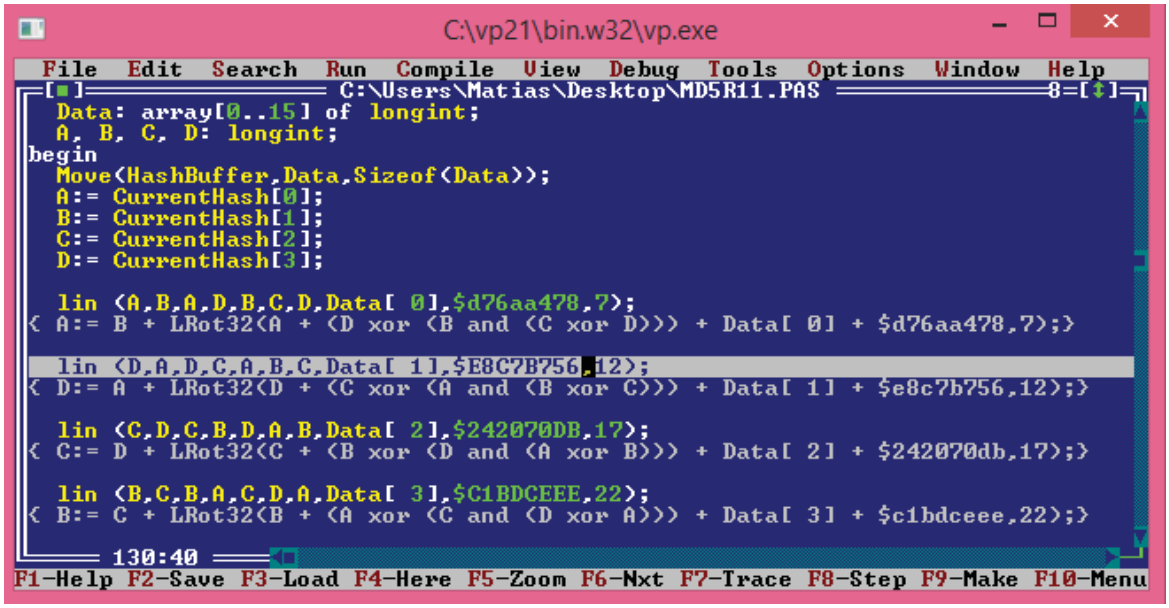


Ilustración 13 segundo ejemplo línea a línea “compress”

- **Step over** (paso a paso rápido) esta opción se activa presionando F8 y es similar a la anterior, pero si la instrucción ejecutada contiene la llamada a una función, entonces la ejecución ejecuta la función completamente en un único paso, evitando tener que ejecutar la función paso a paso. O se debe utilizar cuando se dese depurar una función que se supone correcta, se debe recordar que esta opción ejecuta una función en un único paso [12].

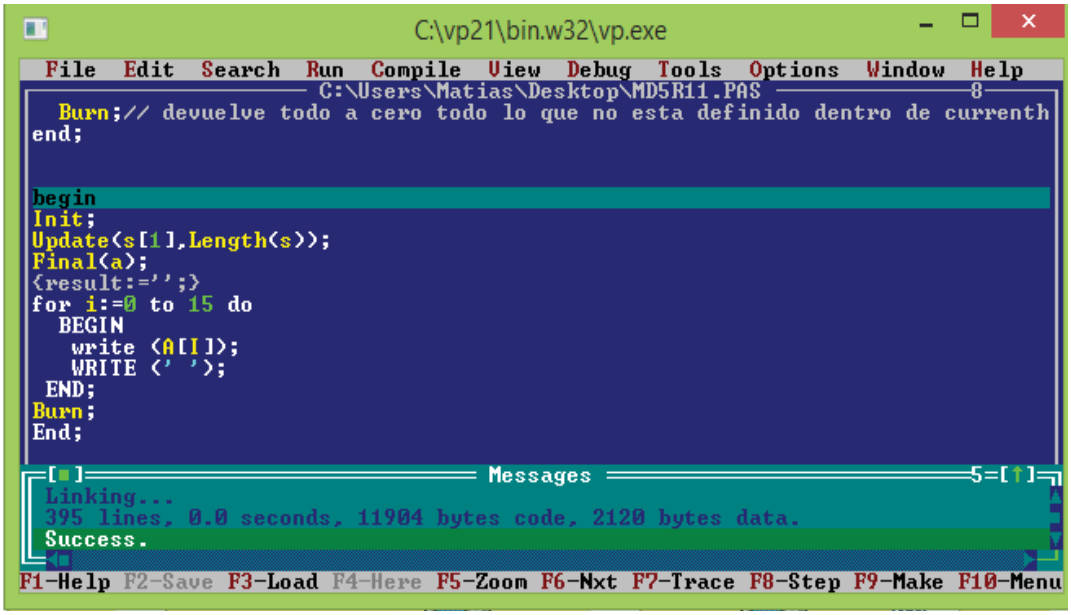


Ilustración 14 ejemplo "step over"

En la ilustración 15 se muestra un ejemplo del cursor apuntando a una función con "step over" se hace ejecuta la función rápidamente y no ingresa a esta línea a línea como la opción anterior como queda explícito en las siguientes ilustraciones.

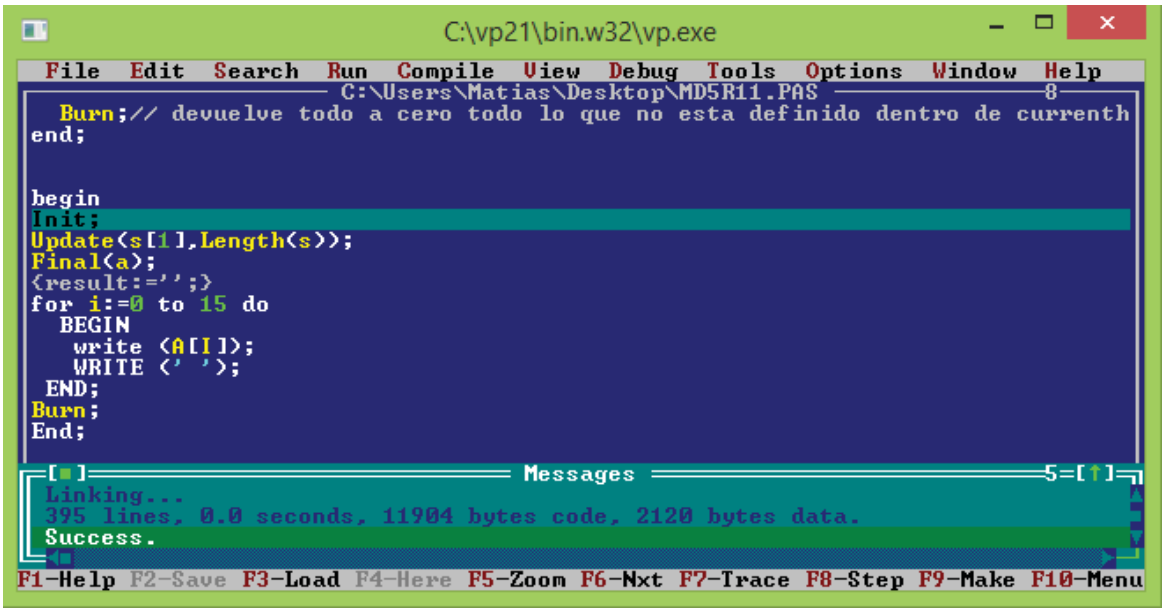


Ilustración 15 ejemplo2 “step over”

En la ilustración 16 se muestra como ejecuta el procedimiento “init” sin entrar en la línea a línea como la opción anterior.

- Execute to: La opción Run (Ctrl-F9) es la más útil y permite ejecutar el programa. Si el programa no está compilado, lo compila previamente y si tiene errores, los visualizará (sin ejecutar el programa, obviamente) en una ventana denominada Message.
- Las demás opciones y el menú Debug son opciones del debugger o depurador de programas y se explican más adelante [12]

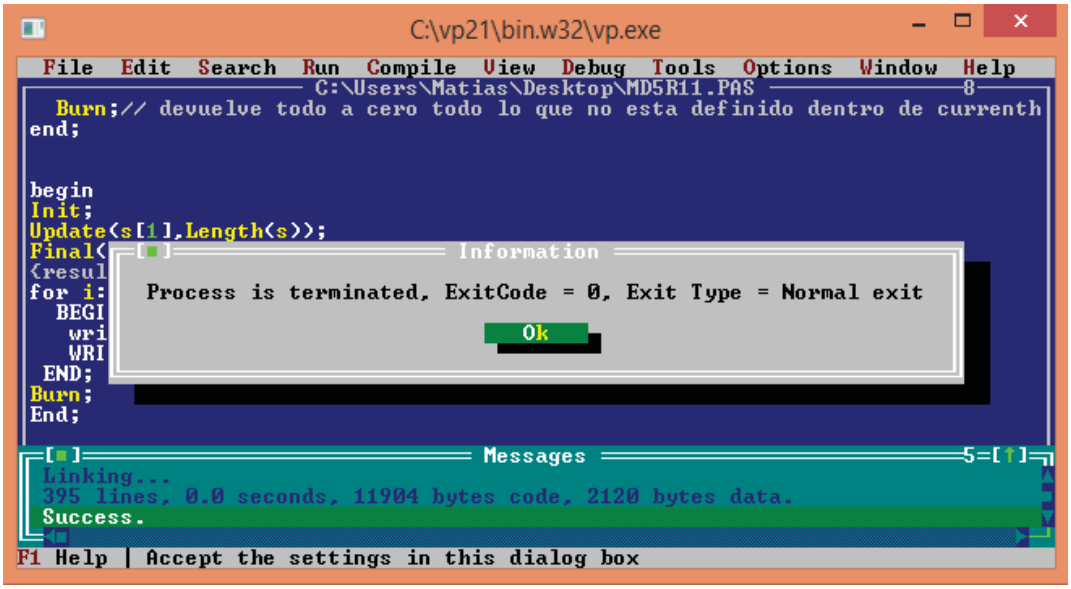


Ilustración 16 Opción “Execute to”

6.6 Compile

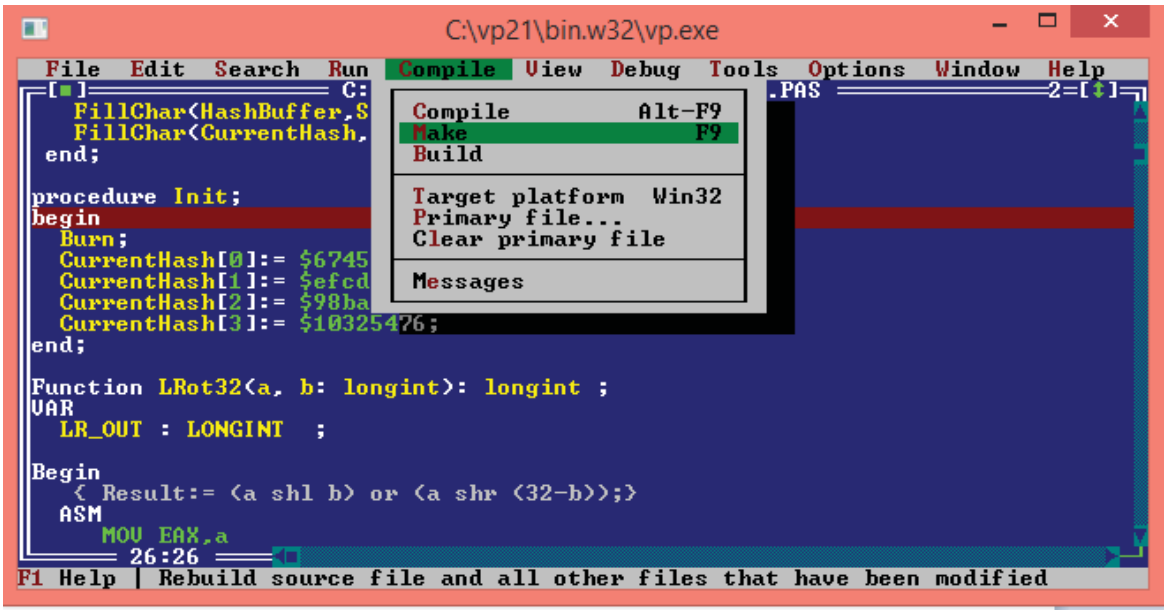


Ilustración 17 Opción “Compile”

- **Compile (Alt-F9)** Son las opciones para compilar el programa sin necesidad de ejecutarlo posteriormente. Se distingue entre compilar (compile) el programa que consiste en unirle las funciones de librerías cuya definición no está incluida en el programa. Al final se generarán ficheros con la extensión. Recuerde que si lo que desea es ejecutar el programa para probar su funcionamiento, la opción Run compila el programa si es necesario (si ha habido alguna modificación), por lo que no es necesario compilar primero y luego ejecutar.

Ilustración 18 Opción “Compile”

6.7 View

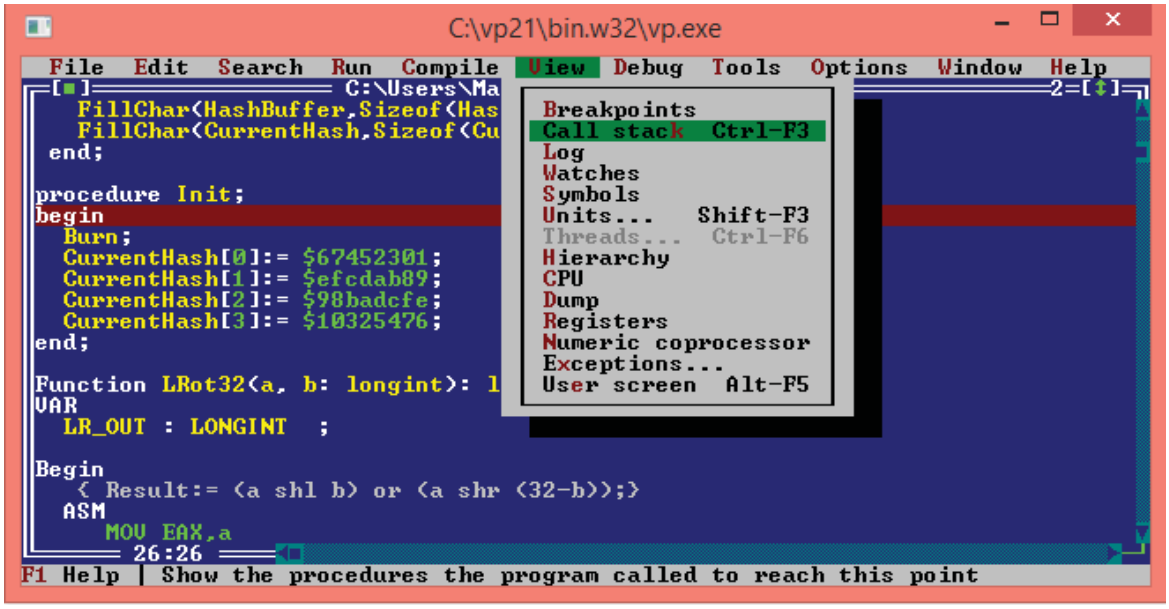


Ilustración 19 Opción “View”

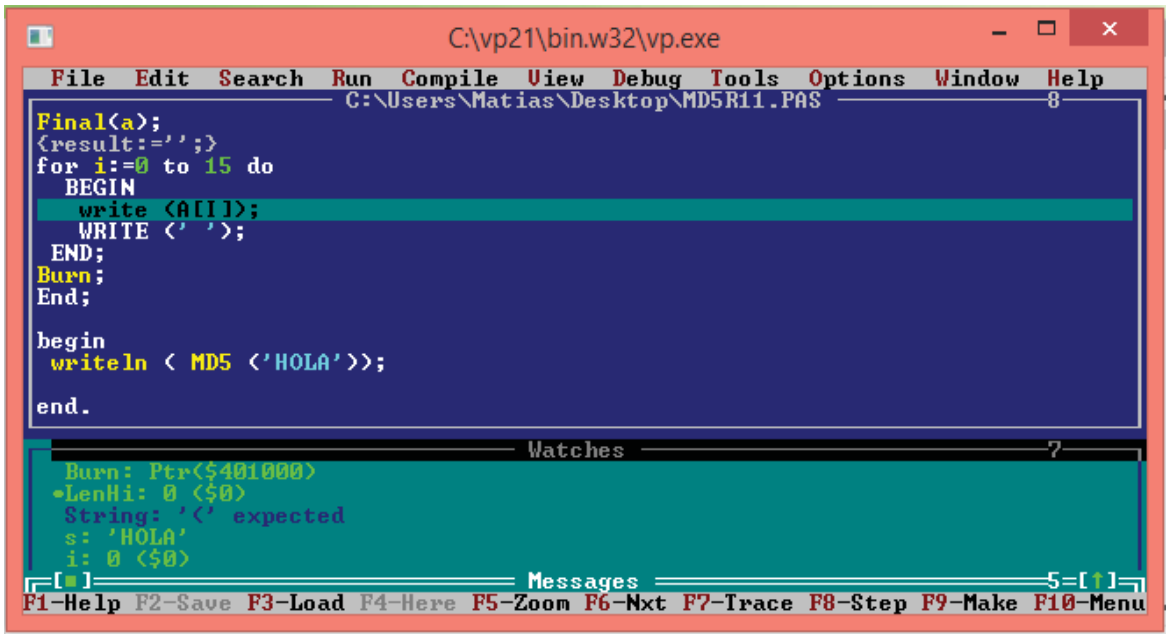


Ilustración 20 Opción "watches"

En esta sección una de las opciones más útiles que permite ver con exactitud los valores que van tomando las variables dentro los procedimientos y funciones como se muestra en la ilustración 21, lo cual permite verificar alguna falla o error o comparar los resultados obtenidos, lo cual le da gran utilidad a la herramienta [12].

6.8 Debug

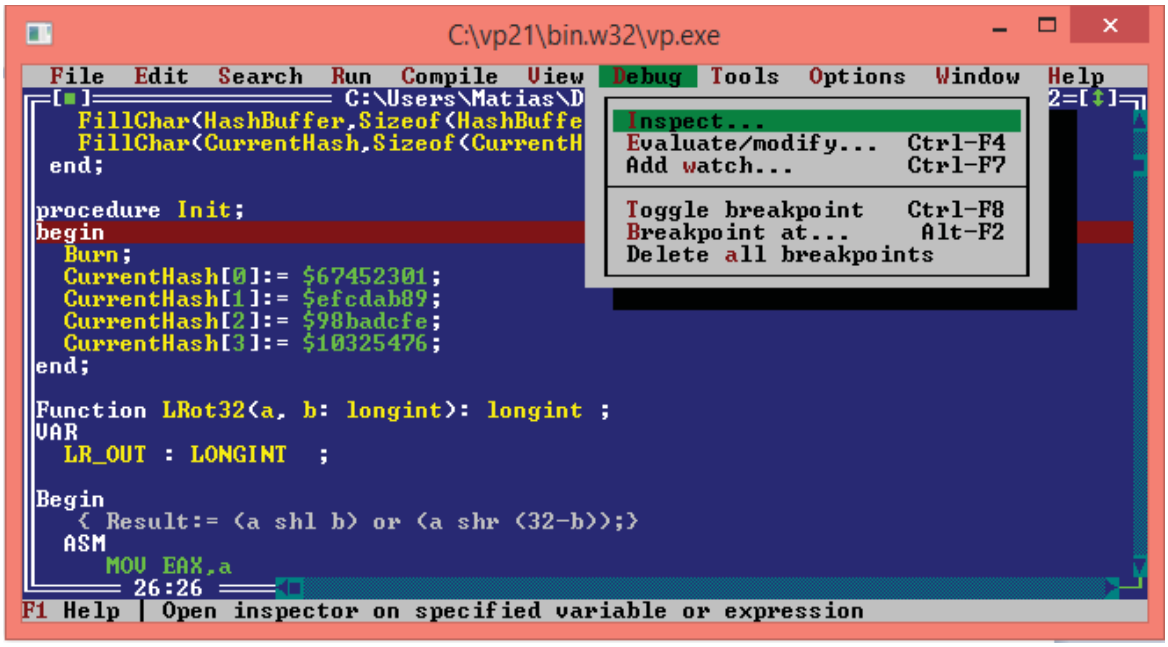


Ilustración 21 Opción "Debug"

Ese menú contiene, básicamente algunas opciones para la depuración de programas. Las más importante son las que se refieren a visualizar el contenido de las variables e incluso modificar su valor.

- **Opción Evaluate/modify Ctrl-F4 (Evaluar/Modificar):** Utiliza esta opción para visualizar el contenido de una variable (o una expresión) con la posibilidad de cambiar su contenido en tiempo de ejecución. [12]

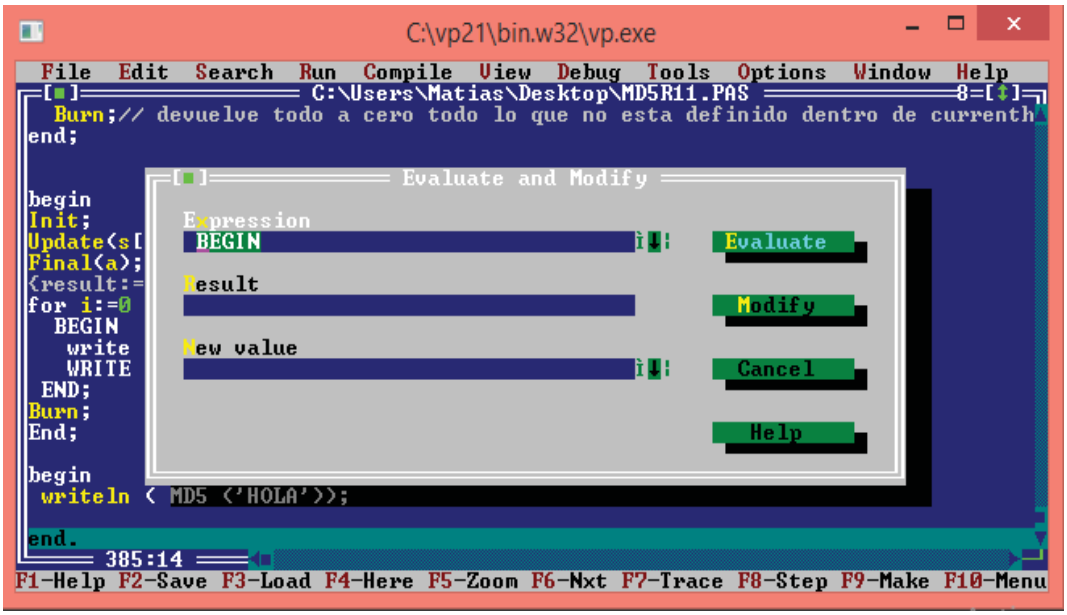


Ilustración 22 Opción "Evaluar/ modificar"

- **Add watch** La opción anterior es útil para cuando queremos evaluar alguna variable o expresión de forma muy puntual. Sin embargo, a veces queremos ver el valor de una variable o expresión de forma continua, para así ir viendo cómo se modifica su valor en cada paso que ejecutamos con el depurador. Si usamos la opción Add Watch Ctrl-F7 abre

una ventana que nos pide que escribamos la variable o expresión que queremos evaluar [12].

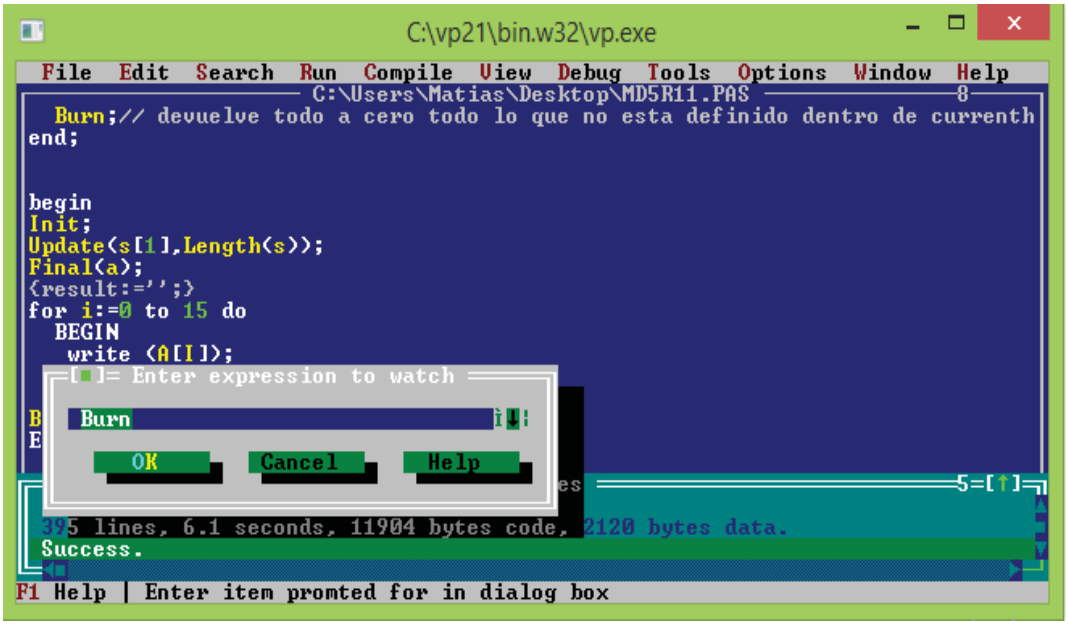


Ilustración 23 Opción “add watch”

7 Transferencia de datos con MD5

7.1 Protocolo TLS

Este protocolo consta de tres sub-protocolos que permiten que los interlocutores lleguen a un acuerdo para los parámetros de seguridad que sean empleados para el nivel de registro, se autentifiquen, parámetros de seguridad negociados y se comuniquen condiciones de error.

Este protocolo es el principal responsable de negociar una sesión que consta de los siguientes ítems:

- Identificador de sesión: Secuencia de bytes aleatoria elegida por el servidor para identificar el estado de una sesión activa o reanudable.
- Certificado del interlocutor: Certificado X.509 v3 del interlocutor. Este elemento puede ser nulo.
- Método de compresión: Algoritmo empleado para comprimir los datos antes de encriptarlos.
- Especificaciones del algoritmo de encriptación: Especifica el algoritmo de encriptación (nulo, DES, etc.) y el algoritmo de firmado (MD5 o SHA). También define atributos criptográficos como el tamaño de la clave de la función de dispersión.
- Secreto principal: Clave secreta de 48 bytes compartida entre el cliente y el servidor.
- Reutilizable: Valor que indica si la sesión puede ser empleada para iniciar nuevas conexiones.

Figura 7-1 Secuencia protocolo TLS (https://revista.seguridad.unam.mx)

7.1.1 Protocolo de cambio de especificaciones criptográficas

Este protocolo marca las transiciones entre distintas estrategias de cifrado. Consta de un mensaje que se encripta y comprime con las especificaciones actuales de la conexión (no las pendientes).

Cuando el destinatario recibe este mensaje la capa de registro copia el estado de lectura pendiente al estado de lectura actual. De forma similar, el emisor cambia su estado de escritura al enviar este mensaje. Este mensaje se envía durante el acuerdo, después de haber acordado los parámetros de seguridad, pero antes de que se envíe el mensaje de verificación finalizada [13].

7.1.2 Protocolo de alerta

Uno de los tipos de mensaje que soporta la capa de registro es el de alerta. Estos mensajes incluyen la severidad de la alerta y una descripción de ésta. Los mensajes de alerta con nivel de fatal provocan la inmediata terminación de la comunicación [13].

Existen distintos tipos de alertas:

- Alerta de cierre. El cliente y el servidor deben saber que la conexión se está cerrando para evitar un ataque de truncado. Cualquiera de los dos puede iniciar el intercambio de mensajes de cierre. Cualquier información recibida después de la alerta de cierre es ignorada.
- Alerta de error. La gestión de errores el protocolo de mutuo acuerdo es muy simple, cuando uno de los interlocutores detecta un error lo envía al otro y, si se trata de un error fatal, cierran la conexión.

7.1.3 Protocolo de mutuo acuerdo

Los parámetros del estado de la sesión son producidos por este protocolo, que opera sobre el protocolo de registro TLS. Cuando un cliente y un servidor empiezan a comunicarse, acuerdan la

versión del protocolo, selección de algoritmos criptográficos, opcionalmente se autentifican mutuamente y emplean algoritmos de clave pública para generar secretos compartidos [14].

El protocolo de mutuo acuerdo consta de los siguientes pasos:

- Intercambio de mensajes de saludo para acordar los algoritmos a emplear, intercambiar valores aleatorios y verificar si es una sesión reanudada [14].
- Intercambiar los parámetros criptográficos necesarios para permitir que el cliente y el servidor acuerden un pre-acuerdo [14].
- Intercambio de certificados e información criptográfica para permitir que cliente y servidor se autentifiquen [14].
- Generar un secreto principal a partir del pre-acuerdo e intercambiar valores aleatorios.
- Proporcionar los parámetros de seguridad a la capa de registro [14].
- Permitir al cliente y al servidor verificar que su interlocutor a calculado los mismos parámetros de seguridad y que el acuerdo se produjo sin alteraciones por parte de un tercero [14].

7.1.4 Protocolo de datos de aplicación

Los mensajes de datos de la aplicación son transportados por la capa de registro y son fragmentados, comprimidos y encriptados basándose en el estado actual de la conexión. Los mensajes se tratan como datos transparentes para la capa de registro [14].

7.2 Certificados X.509 y la autoridad de certificacion

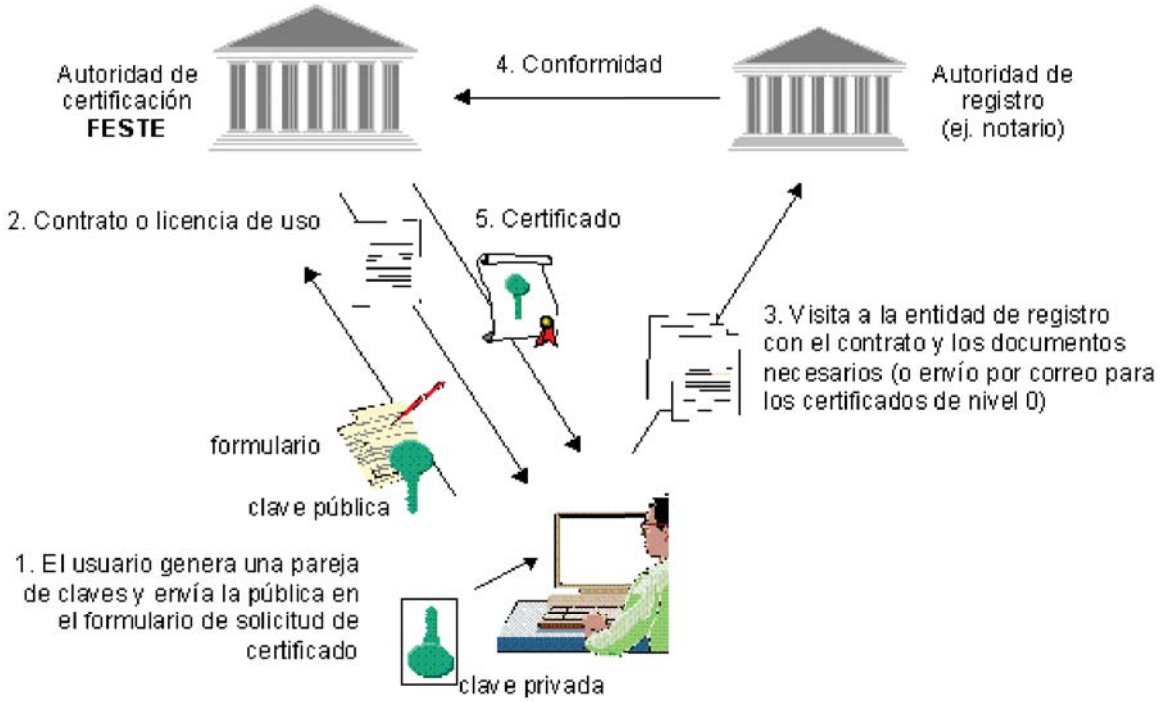


Figura 7-2 Secuencia certificación(www.dit.upm.es)

La creciente demanda para las conexiones encriptadas que garanticen la confidencialidad (nadie más que el receptor legítimo puede ver los contenidos) de los datos transmitidos sobre la red, la veracidad (firma electrónica) e integridad (no es posible modificar el contenido entre los dos puntos finales de las comunicaciones) ha dado lugar a la propagación de los algoritmos de cifrado asimétricos, más conocidos como "algoritmos de clave pública" [14]. A diferencia de cifrado de clave simétrica, un par de claves duales se utilizan en el cifrado de clave pública: lo que se cifra con una, solo se puede descifrar con la otra y viceversa. Una de las dos llaves se llama la clave privada y la otra, clave pública. Como el nombre sugiere, la clave privada es secreta y por lo tanto deben ser cuidadosamente custodiada por su propietario, en lugar de la clave pública que se distribuye a los contactos [14].

Si un usuario "A" quiere enviar información secreta a un usuario "B", entonces "A" debe cifrar los datos con la clave pública de "B" [14]. Donde "B" es el único poseedor de la clave privada correspondiente para el descifrado. Esto resuelve el problema de la confidencialidad [14].

Ahora supongamos que el usuario "A" quiere asegurarse de que los datos recibidos son inequívocamente de "B" y que no se han alterado en el camino: "B" debe calcular el hash de los datos que se envíen y cifrar con la clave privada. Cuando "A" recibe los datos y el HASH cifrado, descifra esta última con la clave pública de" B" y compararlo con el HASH que se calcula sobre los datos recibidos. Si los dos valores HASH son iguales entonces "A" puede estar seguro de la identidad de la "B" y la integridad de los datos. En particular, si los datos enviados por "B" a" A" es un documento electrónico, a continuación, el procedimiento anterior se conoce como "firma digital" [14].

El cifrado asimétrico es tan fuerte como la mayor certeza es que la clave pública pertenece realmente al poseedor del contacto de los mensajes que son enviados, o cuyo origen debe ser autenticada. Lo mejor que se puede hacer es intercambiar personalmente las claves públicas, pero en una organización grande no siempre es posible. Esto resulta en la necesidad de dotar a estas organizaciones con una PKI (Public Key Infrastructure) [14]. PKI se basa en el uso de certificados, es decir, los documentos digitales firmados que contienen los datos de identificación del usuario (o servidor), su clave pública y la identificación de la autoridad de certificación que firma el certificado. Naturalmente, la Autoridad de Certificación (abreviado como CA), que firma el certificado debe ser de confianza y su clave pública distribuidos de una manera segura [14].

Zeroshell es una distribución Linux para servidores y dispositivos integrados destinados a proporcionar los servicios de red principal de una LAN (Red de área local) implementa una CA para la emisión y gestión de certificados digitales X.509 v3 [7]. En particular, hace que sea posible:

- Generar parejas de claves RSA de 512, 1024 y 2048 bits;
- Generar certificados X509.V3 relacionados con los usuarios y servidores;
- Renovar un certificado;
- Exportar un certificado (con o sin la clave privada relacionada) en el PEM, DER y en formato PKCS#12;
- Revocar un certificado antes de su vencimiento;
- Gestión de la CRL (lista de certificados revocados), es decir, la lista de certificados revocados firmada por la autoridad competente;
- Generar la clave privada y el certificado de la entidad emisora de certificados, si se trata de una CA raíz o importar si se trata de una CA intermedia [7];

Con el uso de certificados X.509 v3 y el protocolo SSL (Secure Socket Layer), inicialmente desarrollado por Netscape, pero en la versión 3.0 que fue estandarizado por el IETF y pidió TLS 1.0 (Transport Layer Security), es posible crear túneles de extremo a extremo basado en el nivel de transporte TCP y obtener un nivel de sesión seguro. Ejemplos de protocolos de sesión encapsulado en los túneles SSL incluyen: https (http seguro), imaps (IMAP seguro), telnet (telnet seguro), SMTPS (SMTP seguro), ldaps (LDAP seguro), pop3s (seguro pop3) y nntps (seguro USENET protocolo de transporte). El protocolo TLS no se limita a cifrar los datos que pasan en los túneles, pero cuando se le solicite, sino que también garantiza la autenticidad del cliente y el servidor (autenticación mutua) [15]. A continuación, se puede observar que los algoritmos asimétricos tienen una complejidad computacional mucho mayor en comparación con las simétricas. Por esta razón en el protocolo SSL, gracias a un primer intercambio de datos cifrados asimétrico, clientes y servidores están de acuerdo en un algoritmo simétrico y la tecla correspondiente con la que cifrar el resto de la conversación.

Zeroshell utiliza SSL/TLS para los siguientes fines:

- Para crear sesiones seguras con https entre el huésped y el anfitrión Zeroshell donde se ejecuta utiliza el navegador web para acceder a la interfaz de administración [7];
- En el protocolo 802.1x mediante RADIUS para autenticar las conexiones inalámbricas con EAP-TLS y PEAP;
- Para encapsular las tramas Ethernet en los túneles cifrados, creando así redes VPN de LAN a LAN;
- Para autenticar la conexión IPsec en el que los túneles L2TP son encapsulado para crear L2TP/IPsec Host-to-LAN VPN Road Warrior.

7.3 Protocolo SSH

SSH o Secure Shell, es un protocolo de administración remota que permite a los usuarios controlar y modificar sus servidores remotos a través de Internet. El servicio se creó como un reemplazo seguro para el Telnet sin cifrar y utiliza técnicas criptográficas para garantizar que todas las comunicaciones hacia y desde el servidor remoto sucedan de manera encriptada. Proporciona un mecanismo para autenticar un usuario remoto, transferir entradas desde el cliente al host y retransmitir la salida de vuelta al cliente [16].

7.3.1 Técnicas de cifrado

La ventaja significativa ofrecida por SSH sobre sus predecesores, es el uso del cifrado para asegurar la transferencia segura de información entre el host y el cliente. Host se refiere al servidor remoto al que está intentando acceder, mientras que el cliente es el equipo que está utilizando para acceder al host [17]. Hay tres tecnologías de cifrado diferentes utilizadas por SSH:

- Encriptación simétrica
- Encriptación asimétrica
- Hashing.

7.3.2 SSH con estas téccnicas de cifrado

La forma en que funciona SSH es mediante el uso de un modelo cliente-servidor para permitir la autenticación de dos sistemas remotos y el cifrado de los datos que pasa entre ellos [16].

SSH opera en el puerto TCP 22 de forma predeterminada (aunque esto se puede cambiar si es necesario). El host (servidor) escucha en el puerto 22 (o cualquier otro puerto SSH asignado) para las conexiones entrantes. Organiza la conexión segura mediante la autenticación del cliente y la apertura del entorno de shell correcto si la verificación tiene éxito [16].

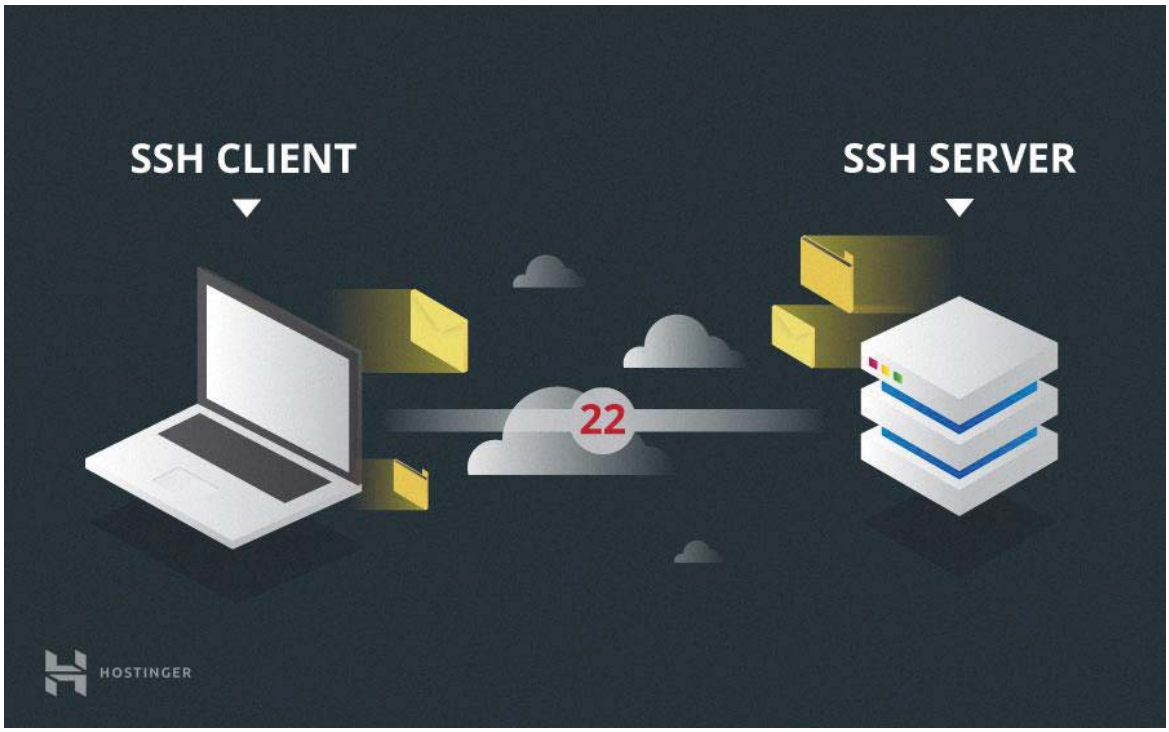


Figura 7-3 Representación protocolo SSH(www.hostinger.es)

El cliente debe iniciar la conexión SSH iniciando el protocolo TCP con el servidor, asegurando una conexión simétrica segura, verificando si la identidad mostrada por el servidor coincide con

los registros anteriores (normalmente grabados en un archivo de almacén de claves RSA) y presenta las credenciales de usuario necesarias para autenticar la conexión [18].

Hay dos etapas para establecer una conexión: primero, ambos sistemas deben acordar estándares de encriptación para proteger futuras comunicaciones, y segundo, el usuario debe autenticarse. Si las credenciales coinciden, se concede acceso al usuario [18].

7.3.3 Negociación de cifrado de sesión

Cuando un cliente intenta conectarse al servidor a través de TCP, el servidor presenta los protocolos de cifrado y las versiones respectivas que soporta. Si el cliente tiene un parecido similar de protocolo y versión, se alcanza un acuerdo y se inicia la conexión con el protocolo aceptado. El servidor también utiliza una clave pública asimétrica que el cliente puede utilizar para verificar la autenticidad del host [15].

Una vez que esto se establece, las dos partes usan lo que se conoce como Algoritmo de Intercambio de Claves Diffie-Hellman para crear una clave simétrica. Este algoritmo permite que tanto el cliente como el servidor lleguen a una clave de cifrado compartida que se utilizará en adelante para cifrar toda la sesión de comunicación [15].

Aquí es como el algoritmo trabaja en un nivel muy básico:

- Tanto el cliente como el servidor coinciden en un número primo muy grande, que por supuesto no tiene ningún factor en común. Este valor de número primo también se conoce como el valor de la semilla [15].
- Luego, las dos partes acuerdan un mecanismo de cifrado común para generar otro conjunto de valores manipulando los valores de semilla de una manera algorítmica específica. Estos mecanismos, también conocidos como generadores de cifrado, realizan grandes operaciones sobre la semilla. Un ejemplo de dicho generador es AES (Advanced Encryption Standard) [15].
- Ambas partes generan independientemente otro número primo. Esto se utiliza como una clave privada secreta para la interacción [15].
- Esta clave privada recién generada, con el número compartido y el algoritmo de cifrado (por ejemplo, AES), se utiliza para calcular una clave pública que se distribuye a la otra computadora [15].
- A continuación, las partes utilizan su clave privada personal, la clave pública compartida de la otra máquina y el número primo original para crear una clave compartida final. Esta clave se calcula de forma independiente por ambos equipos, pero creará la misma clave de cifrado en ambos lados [15].
- Ahora que ambas partes tienen una clave compartida, pueden encriptar simétricamente toda la sesión SSH. La misma clave se puede utilizar para cifrar y descifrar mensajes (leer: sección sobre cifrado simétrico) [15].

7.3.4 Auntentificacion del usuario

La etapa final antes de que se conceda al usuario acceso al servidor es la autenticación de sus credenciales. Para ello, la mayoría de los usuarios de SSH utilizan una contraseña. Se le pide al

usuario que introduzca el nombre de usuario, seguido de la contraseña. Estas credenciales pasan con seguridad a través del túnel cifrado simétricamente, así que no hay ninguna posibilidad de que sean capturadas por un tercero [15].

Aunque las contraseñas se cifran, todavía no se recomienda usar contraseñas para conexiones seguras. Esto se debe a que muchos robots pueden simplemente activar las contraseñas fáciles o predeterminadas y obtener acceso a su cuenta. En su lugar, la alternativa recomendada es SSH Key Pairs [15].

Se trata de un conjunto de claves asimétricas utilizadas para autenticar al usuario sin necesidad de introducir ninguna contraseña [15].

7.4 Intercambio de archivos entre instituciones

Se pone a disposición de las instituciones autorizadas su repositorio transitorio para el intercambio de archivos entre ellas, para ser más clara esta explicación se tomará la institución Superintendencia que no será ni el origen ni el destino del archivo [15]. Para este caso, solo se hará cargo de traspasar los archivos desde una institución a otra según corresponda como tampoco se hace cargo de la validación y verificación del envió entre instituciones [15].

A continuación, un ejemplo donde se muestra en la tabla las casillas para recepción y envíos que deberá utilizar cada institución según la naturaleza de la transmisión:

Institución	Directorios	Descripción
Administradoras de Fondos de Pensiones	desdesp haciasp	TEA con la Superintendencia
	desdecsv haciacsv	TEA con las Compañías de Seguros de Vida
	desdeips haciaips	TEA con el Instituto de Previsión Social
Administradora de Fondos de Cesantía	desdesp haciasp	TEA con la Superintendencia
Instituto de Previsión Social	desdesp haciasp	TEA con la Superintendencia
	desdeafp haciaafp	TEA con las Administradoras de Fondos de Pensiones
	desdecsv haciacsv	TEA con las Compañías de Seguros de Vida
Compañías de Seguros de Vida	desdesp haciasp	TEA con la Superintendencia
	desdeafp haciaafp	TEA con las Administradoras de Fondos de Pensiones
	desdeips haciaips	TEA con el Instituto de Previsión Social
Otras instituciones	desdesp haciasp	TEA con la Superintendencia

Tabla 7-1 casillas para la recepción y envíos de archivos

7.5 Generación de MD5 para archivos a transmitir

El archivo MD5 actúa como "resumen digital" de un informe específico y permite comprobar si el archivo de datos asociado ha sido modificado o dañado, cumpliendo además la función de indicador del estado de cada envío. El resultado que genera el HASH para un archivo zip debe guardarse en un segundo archivo que contenga el nombre del archivo al cual se le calculó su HASH y el valor del MD5, utilizándose para ello el siguiente formato en una sola línea [18].

Información del Campo	Tamaño	Contenido o Formato
MD5	X(32)	El MD5 calculado
Filler	X(01)	En blanco
Asterisco	X(01)	*
Nombre archivo de datos	Indefinido	Nombre del archivo al cual se le calculó el MD5

Tabla 7-2 datos a transmitir

Por ejemplo, el contenido del archivo con el MD5 para el archivo "pre030721_030722_1016.zip" podría ser:

- be1c2b0d0419eb3abb05d23230f64e1f *pre030721_030722_1016.zip
  La generación del MD5 debe ser calculado utilizando la "opción binaria". En la transmisión de archivos se debe enviar en cada oportunidad el par de archivos: archivo de datos empaquetados en formato "zip" y archivo que contiene el cálculo del MD5 del archivo de datos. Con ellos, la institución que recibe el archivo debe verificar si el archivo ha sido modificado o dañado. Para cada archivo de datos, debe generarse su correspondiente archivo con el cálculo del MD5 el cual debe ser transmitido junto a él, según se indica en el siguiente punto [16].

7.6 Sincronización de envío y recepción de archivos

El orden estricto de transmisión de archivos es el siguiente:

- Transmitir el archivo de datos
- Transmitir el archivo con el cálculo del MD5 asociado.

Respecto de la recepción de archivos, el primer paso consiste en leer la "casilla" de recepción y obtener el listado de todos los archivos con el cálculo del MD5, esto quiere decir que todos los archivos con transmisión exitosa. Luego para cada uno de los archivos con el cálculo del MD5 se debe recuperar el par de archivos asociados: datos y MD5. Por ello, el orden estricto de recepción de archivos es:

- Obtener el listado con MD5
- Para cada uno de los archivos MD5 recuperar el respectivo archivo de datos
- Recuperar el archivo MD5 asociado al segundo paso.

7.7 Revisión de consistencia

Una vez recuperados el par de archivos (dato, MD5), la institución receptora debe revisar la consistencia de los archivos recibidos a través de la comprobación del cálculo MD5 indicando en el archivo con MD5 y su respectivo archivo de datos [16].

8 Programación de algoritmo MD5

8.1 Declaración de variables

Dentro de esta sección declararemos las variables globales y locales dentro de una función llamada MD5 que será la función central del proyecto, dentro de esta función se declaran otras funciones y procedimientos que poseen sus propias variables locales.

Listado 8-1 Declaración de variables

```
1  Program test_md5 ;
2
3
4  Var Clave :String ;
5
6  Function md5(s:string):string;
   var a:array[0..15] of byte;
7      i:integer;

8   LenHi, LenLo: longint ;
    Index: longint;
9   HashBuffer: array[0..63] of byte;
    CurrentHash: array[0..3] of longint ;
```

Dentro del listado 1-1 se declara la variable global “Clave” de carácter String que nos indica que todo lo que posee dentro son caracteres y no un valor numérico. Esto asegura que al final lo que se va a codificar sea una palabra de longitud arbitraria que posea las propiedades de una función hash.

También se declaran variables locales dentro de la función” md5”, estas variables son “a” de tipo array de 16 posiciones en su tamaño donde cada posición corresponde a un byte, junto con esta,

se crearán las variables “i” la cual almacenará números enteros y las variables “LenHi”, “LenLo” y “Index” de carácter longint.

La variable “HashBuffer” cumple el mismo objetivo que la variable “a”, pero la cantidad de posiciones en su tamaño es de 64. Por otro lado “CurrentHash” es un array de 4 posiciones, pero en cada una de esta almacena longint.

Señala que “md5” como la función principal del programa la cual incluirán todos los procedimientos y funciones necesaria para nuestro algoritmo, las cuales tienen sus variables locales que muchas veces pueden obtener el mismo nombre pero que actúan de forma distinta en cada una de estas estructuras, por lo cual se explicaran esas variables junto a sus procedimientos y funciones.

8.2 Funciones y procedimientos

Se explicará el objetivo que cumple cada uno de estos en el orden lógico que está establecido en el programa, donde todas estas funciones y procedimientos se empiezan a ejecutar llamando a la función “md5”.

Listado 8-2 Procedimientos “Burn” y “Init”

```
procedure Burn;
 begin
   LenHi:= 0; LenLo:= 0;
   Index:= 0;
   FillChar(HashBuffer,Sizeof(HashBuffer),0);
   FillChar(CurrentHash,Sizeof(CurrentHash),0);
 end;


procedure Init;
begin
  Burn;
  CurrentHash[0]:= $67452301;
  CurrentHash[1]:= $efcdab89;
  CurrentHash[2]:= $98badcfe;
  CurrentHash[3]:= $10325476;
end;
```

Este procedimiento cumple con el objetivo de darle el valor cero a las variables “LenHi”, “LenLo”, “Index”, por otro lado a través del FillChar se rellenan con ceros las posiciones de la variable “HashBuffer” donde el “Sizeof” devuelve el tamaño de la variable indicada dentro de este en byte que s este caso corresponde al tamaño de “HashBuffer”, lo mismo para la variable CurrentHash que se rellenan los 4 espacios del array con ceros de tipo longint que equivalen a 4 bytes para cada espacio.

En el procedimiento “Init” lo primero que se hace es llamar a “Burn” que lleva todos los valores anteriores a cero y con esto se asignan los valores iníciales del buffer MD a cada uno de los

espacios de la variable “currentHash” mientras todo lo demás sigue en cero, así se puede inicializar y cumplir con uno de los 5 pasos claves del algoritmo MD5.

Listado 8-3 Procedimiento “Update”

```
procedure Update(const Buffer; Size: longint);
var
  PBuf: ^byte;
begin
  Inc(LenHi,Size shr 29);
  Inc(LenLo,Size*8);
  if LenLo< (Size*8) then
    Inc(LenHi);

  PBuf:= @Buffer;
  while Size> 0 do
  begin
    if (Sizeof(HashBuffer)-Index)<= Size then
    begin
      Move(PBuf^,HashBuffer[Index],Sizeof(HashBuffer)-
Index);
      Dec(Size,Sizeof(HashBuffer)-Index);
      Inc(PBuf,Sizeof(HashBuffer)-Index);
      Compress;
    end
    else
    begin
      Move(PBuf^,HashBuffer[Index],Size);
      Inc(Index,Size);
      Size:= 0;
    end;
  end;
end;
```

Este es un procedimiento el cual es llamado en el bucle del programa ingresando un string en md5 que se almacenara en la variable “s”, se define como “Update (s [1], Length(s))” donde la constante “buffer” toma el valor del segundo espacio ingresado en la variable “s” definida en md5, y la variable “Size” de tipo longint almacenara el valor del número de caracteres totales que tiene el String almacenado en “s”.

Luego viene la definición de variables donde se encuentra “PBuf” es un puntero que contiene la dirección de espacio de memoria donde hay almacenado una variable del tipo byte, la cual contiene el primer el primer carácter de la matriz “s” dentro del bucle principal del procedimiento se ejecuta “Inc (LenHi, Size shr 29)” incrementa el valor de la variable “LenHi” en el valor de “Size” y con el comando shr lo desplaza 29 espacios en binario, luego para “LenLo” hay un incremento del valor almacenado en “Size” multiplicado por 8, ejecutado este paso se ingresa a una condición “if” que nos indica que si la variable “LenLo” es menor a la variable ”Size” multiplicada por 8, entonces se ejecuta una accion que incrementa en 1 el valor de la variable “LenHi”, después se almacena “PBuf” en la dirección de espacio de memoria de buffer que equivale a la segunda posición del string en la variable “s”, terminada esta acción se ingresa a una nueva condición

"While" que nos indica que mientras el valor almacenado en la variable "Size" sea mayor a cero se ejecutara un bucle dentro de esta condición. Dentro del "While" se encuentra una condición "if" que apunta a que si el tamaño del array "HashBuffer" que equivale a 16 menos el valor de "Index" que en el procedimiento anterior tomo el valor cero es menor al tamaño de "Size" se ingresa a nuevo bucle dentro de esta condición, es decir si el tamaño del string almacenado en "s" es mayor a 16 se ingresa al bucle del "if" sino entra a la condición else, si "Size" es mayor a 16 la primera instrucción dentro del "if" es mover lo almacenado en la dirección del espacio de memoria "^PBuf" al array "HashBuffer" desde la posición de "index" y que lo almacenado es del tamaño de "HashBuffer" menos cero que es el valor ultimo valor de la variable "index" asi se rellena "Hashbuffer"con los valores desde la segunda posición del buffer en adelante hasta completar los 16 espacios de byte, luego se disminuye en "HashBuffer" menos "index" la variable "Size" para que asi después de ejecutarse todo el bucle se pueda salir en algún momento de la condición "While", luego hay un incremento de "PBuf" del mismo valor antes de llamar al procedimiento compress el cual se ejecutará la cantidad de veces que "Size" sea mayor a cero. Se finaliza el bucle de la última condición. En el caso que no se cumpla la última condición se ingresará a la condición "else", donde "HashBuffer" es un array de 16 byte yes el destino donde van a llegar los datos de "PBuf^" que apunta al espacio de memoria de "Buffer" y se rellena desde la posición del destino con valor igual a "index" en este caso igual a cero y almacena valores igual a "Size" desde "s[1]" hacia adelante que es el espacio de memoria de donde se apunta en "Bufer", por último se incrementa en el valor de "Size" y se le da el valor de cero a la variable "Size", se finalizan los bucles de las condiciones anteriores y se termina el procedimiento.

Listado 8-4 Procedimiento "compress"

```
procedure Compress;
var
  Data: array[0..15] of longint;
  A, B, C, D: longint;
begin
  Move(HashBuffer,Data,Sizeof(Data));
  A:= CurrentHash[0];
  B:= CurrentHash[1];
  C:= CurrentHash[2];
  D:= CurrentHash[3];

  OP1 (A,B,A,D,B,C,D,Data[ 0],$d76aa478,7);
{ A:= B + LRot32(A + (D xor (B and (C xor D))) + Data[
0] + $d76aa478,7);}

  OP1 (D,A,D,C,A,B,C,Data[ 1],$E8C7B756,12);
{ D:= A + LRot32(D + (C xor (A and (B xor C))) + Data[
1] + $e8c7b756,12);}

  OP1 (C,D,C,B,D,A,B,Data[ 2],$242070DB,17);
{ C:= D + LRot32(C + (B xor (D and (A xor B))) + Data[
2] + $242070db,17);}

  OP1 (B,C,B,A,C,D,A,Data[ 3],$C1BDCEEE,22);
{ B:= C + LRot32(B + (A xor (C and (D xor A))) + Data[
3] + $c1bdceee,22);}
```

```
  OP1 (A,B,A,D,B,C,D,Data[ 4],$F57C0FAF,7);
{  A:= B + LRot32(A + (D xor (B and (C xor D))) + Data[
4] + $f57c0faf,7);}

  OP1 (D,A,D,C,A,B,C,Data[ 5],$4787C62A,12);
{  D:= A + LRot32(D + (C xor (A and (B xor C))) + Data[
5] + $4787c62a,12);}

  OP1 (C,D,C,B,D,A,B,Data[ 6],$A8304613,17);
{  C:= D + LRot32(C + (B xor (D and (A xor B))) + Data[
6] + $a8304613,17);}

  OP1 (B,C,B,A,C,D,A,Data[ 7],$FD469501,22);
{  B:= C + LRot32(B + (A xor (C and (D xor A))) + Data[
7] + $fd469501,22);}

  OP1 (A,B,A,D,B,C,D,Data[ 8],$698098D8,7);
{  A:= B + LRot32(A + (D xor (B and (C xor D))) + Data[
8] + $698098d8,7);}

  OP1 (D,A,D,C,A,B,C,Data[ 9],$8B44F7AF,12);
{  D:= A + LRot32(D + (C xor (A and (B xor C))) + Data[
9] + $8b44f7af,12);}

  OP1 (C,D,C,B,D,A,B,Data[ 10],$FFFF5BB1,17);
{  C:= D + LRot32(C + (B xor (D and (A xor B))) +
Data[10] + $ffff5bb1,17);}

  OP1 (B,C,B,A,C,D,A,Data[11],$895CD7BE,22);
{  B:= C + LRot32(B + (A xor (C and (D xor A))) +
Data[11] + $895cd7be,22);}

  OP1 (A,B,A,D,B,C,D,Data[ 12],$6B901122,7);
{  A:= B + LRot32(A + (D xor (B and (C xor D))) +
Data[12] + $6b901122,7);}

  OP1 (D,A,D,C,A,B,C,Data[ 13],$FD987193,12);
{  D:= A + LRot32(D + (C xor (A and (B xor C))) +
Data[13] + $fd987193,12);}

  OP1 (C,D,C,B,D,A,B,Data[ 14],$A679438E,17);
{  C:= D + LRot32(C + (B xor (D and (A xor B))) +
Data[14] + $a679438e,17);}

  OP1 (B,C,B,A,C,D,A,Data[ 15],$49B40821,22);
{  B:= C + LRot32(B + (A xor (C and (D xor A))) +
Data[15] + $49b40821,22);}




  OP1 (A,B,A,C,D,B,C,Data[ 1],$F61E2562,5);        {  ok
}
{  A:= B + LRot32(A + (C xor (D and (B xor C))) + Data[
1] + $f61e2562,5);}
```

```
  OP1 (D,A,D,B,C,A,B,Data[ 6],$C040B340,9);
{  D:= A + LRot32(D + (B xor (C and (A xor B))) + Data[
6] + $c040b340,9);}

  OP1 (C,D,C,A,B,D,A,Data[ 11],$265E5A51,14);
{  C:= D + LRot32(C + (A xor (B and (D xor A))) +
Data[11] + $265e5a51,14);}

  OP1 (B,C,B,D,A,C,D,Data[ 0],$E9B6C7AA,20);
{  B:= C + LRot32(B + (D xor (A and (C xor D))) + Data[
0] + $e9b6c7aa,20);}

  OP1 (A,B,A,C,D,B,C,Data[ 5],$D62F105D,5);
{  A:= B + LRot32(A + (C xor (D and (B xor C))) + Data[
5] + $d62f105d,5);}

  OP1 (D,A,D,B,C,A,B,Data[ 10],$02441453,9);
{  D:= A + LRot32(D + (B xor (C and (A xor B))) +
Data[10] + $02441453,9);}

  OP1 (C,D,C,A,B,D,A,Data[ 15],$D8A1E681,14);
{  C:= D + LRot32(C + (A xor (B and (D xor A))) +
Data[15] + $d8a1e681,14);}

  OP1 (B,C,B,D,A,C,D,Data[ 4],$E7D3FBC8,20);
{  B:= C + LRot32(B + (D xor (A and (C xor D))) + Data[
4] + $e7d3fbc8,20);}

  OP1 (A,B,A,C,D,B,C,Data[ 8],$21E1CDE6,5);       {  not
ok  }
{  A:= B + LRot32(A + (C xor (D and (B xor C))) + Data[
9] + $21e1cde6,5);}

  OP1 (D,A,D,B,C,A,B,Data[ 14],$C33707D6,9);
{  D:= A + LRot32(D + (B xor (C and (A xor B))) +
Data[14] + $c33707d6,9);}

  OP1 (C,D,C,A,B,D,A,Data[ 3],$F4D50D87,14);
{  C:= D + LRot32(C + (A xor (B and (D xor A))) + Data[
3] + $f4d50d87,14);}

  OP1 (B,C,B,D,A,C,D,Data[ 8],$455ª14ED,20);
{  B:= C + LRot32(B + (D xor (A and (C xor D))) + Data[
8] + $455ª14ed,20);}


  OP1 (A,B,A,C,D,B,C,Data[ 13],$A9E3E905,5);
{  A:= B + LRot32(A + (C xor (D and (B xor C))) +
Data[13] + $a9e3e905,5);}


  OP1 (D,A,D,B,C,A,B,Data[ 2],$FCEFA3F8,9);
{  D:= A + LRot32(D + (B xor (C and (A xor B))) + Data[
2] + $fcefa3f8,9);}

  OP1 (C,D,C,A,B,D,A,Data[ 7],$676F02D9,14);
```

```
{  C:= D + LRot32(C + (A xor (B and (D xor A))) + Data[
7] + $676f02d9,14);}

  OP1 (B,C,B,D,A,C,D,Data[ 12],$8D2A4C8A,20);
{  B:= C + LRot32(B + (D xor (A and (C xor D))) +
Data[12] + $8d2a4c8a,20);}




  OP2 (A,B,A,B,C,D,Data[5],$fffa3942,4);
{ A:= B + LRot32(A + (B xor C xor D) + Data[ 5] +
$fffa3942,4);}

  OP2 (D,A,D,A,B,C,Data[8],$8771f681,11);
{ D:= A + LRot32(D + (A xor B xor C) + Data[ 8] +
$8771f681,11);}

  OP2 (C,D,C,D,A,B,Data[11],$6d9d6122,16);
{ C:= D + LRot32(C + (D xor A xor B) + Data[11] +
$6d9d6122,16);}

  OP2 (B,C,B,C,D,A,Data[14],$fde5380c,23);
{ B:= C + LRot32(B + (C xor D xor A) + Data[14] +
$fde5380c,23);}

  OP2 (A,B,A,B,C,D,Data[1],$a4beea44,4);
{ A:= B + LRot32(A + (B xor C xor D) + Data[ 1] +
$a4beea44,4);}

  OP2 (D,A,D,A,B,C,Data[4],$4bdecfa9,11);
  { D:= A + LRot32(D + (A xor B xor C) + Data[ 4] +
$4bdecfa9,11);}

  OP2 (C,D,C,D,A,B,Data[7],$f6bb4b60,16);
  { C:= D + LRot32(C + (D xor A xor B) + Data[ 7] +
$f6bb4b60,16);}

  OP2 (B,C,B,C,D,A,Data[10],$bebfbc70,23);
 { B:= C + LRot32(B + (C xor D xor A) + Data[10] +
$bebfbc70,23);}

  OP2 (A,B,A,B,C,D,Data[13],$289b7ec6,4);
 { A:= B + LRot32(A + (B xor C xor D) + Data[13] +
$289b7ec6,4);}

  OP2 (D,A,D,A,B,C,Data[0],$eaa127fa,11);
 { D:= A + LRot32(D + (A xor B xor C) + Data[ 0] +
$eaa127fa,11);}

  OP2 (C,D,C,D,A,B,Data[3],$d4ef3085,16);
 { C:= D + LRot32(C + (D xor A xor B) + Data[ 3] +
$d4ef3085,16);}

  OP2 (B,C,B,C,D,A,Data[6],$04881d05,23);
 { B:= C + LRot32(B + (C xor D xor A) + Data[ 6] +
$04881d05,23);}
```

```
  OP2 (A,B,A,B,C,D,Data[9],$d9d4d039,4);
 { A:= B + LRot32(A + (B xor C xor D) + Data[ 9] +
$d9d4d039,4);}

  OP2 (D,A,D,A,B,C,Data[12],$e6db99e5,11);
 { D:= A + LRot32(D + (A xor B xor C) + Data[12] +
$e6db99e5,11);}

  OP2 (C,D,C,D,A,B,Data[8],$1fa27cf8,16);
 { C:= D + LRot32(C + (D xor A xor B) + Data[15] +
$1fa27cf8,16);}

  OP2 (B,C,B,C,D,A,Data[2],$c4ac5665,23);
 { B:= C + LRot32(B + (C xor D xor A) + Data[ 2] +
$c4ac5665,23);}



  OP3 (A,B,A,C,B,D,Data[0],$f4292244,6);
 { A:= B + LRot32(A + (C xor (B or (not D))) + Data[ 0]
+ $f4292244,6);}

  OP3 (D,A,D,B,A,C,Data[7],$432aff97,10);
 { D:= A + LRot32(D + (B xor (A or (not C))) + Data[ 7]
+ $432aff97,10);}

  OP3 (C,D,C,A,D,B,Data[14],$ab9423a7,15);
 { C:= D + LRot32(C + (A xor (D or (not B))) + Data[14]
+ $ab9423a7,15);}

  OP3 (B,C,B,D,C,A,Data[5],$fc93a039,21);
 { B:= C + LRot32(B + (D xor (C or (not A))) + Data[ 5]
+ $fc93a039,21);}

  OP3 (A,B,A,C,B,D,Data[12],$655b59c3,6);
 { A:= B + LRot32(A + (C xor (B or (not D))) + Data[12]
+ $655b59c3,6); }

  OP3 (D,A,D,B,A,C,Data[3],$8f0ccc92,10);
 { D:= A + LRot32(D + (B xor (A or (not C))) + Data[ 3]
+ $8f0ccc92,10);}

  OP3 (C,D,C,A,D,B,Data[10],$ffEFF47d,15);
 { C:= D + LRot32(C + (A xor (D or (not B))) + Data[10]
+ $ffeff47d,15);}

  OP3 (B,C,B,D,C,A,Data[1],$85845dd1,21);
 { B:= C + LRot32(B + (D xor (C or (not A))) + Data[ 1]
+ $85845dd1,21);}

  OP3 (A,B,A,C,B,D,Data[8],$6fa87e4f,6);
 { A:= B + LRot32(A + (C xor (B or (not D))) + Data[ 8]
+ $6fa87e4f,6); }

  OP3 (D,A,D,B,A,C,Data[15],$fe2ce6e0,10);
```

```
        { D:= A + LRot32(D + (B xor (A or (not C))) + Data[15]
      + $fe2ce6e0,10);}

        OP3 (C,D,C,A,D,B,Data[6],$a3014314,15);
       { C:= D + LRot32(C + (A xor (D or (not B))) + Data[ 6]
      + $a3014314,15);}

        OP3 (B,C,B,D,C,A,Data[13],$4e0811A1,21);
       { B:= C + LRot32(B + (D xor (C or (not A))) + Data[13]
      + $4e0811a1,21);}

        OP3 (A,B,A,C,B,D,Data[4],$f7537e82,6);
       { A:= B + LRot32(A + (C xor (B or (not D))) + Data[ 4]
      + $f7537e82,6); }

        OP3 (D,A,D,B,A,C,Data[11],$bd3af235,10);
       { D:= A + LRot32(D + (B xor (A or (not C))) + Data[11]
      + $bd3af235,10);}

        OP3 (C,D,C,A,D,B,Data[2],$2ad7d2bb,15);
       { C:= D + LRot32(C + (A xor (D or (not B))) + Data[ 2]
      + $2ad7d2bb,15);}

        OP3 (B,C,B,D,C,A,Data[9],$eb86d391,21);
       { B:= C + LRot32(B + (D xor (C or (not A))) + Data[ 9]
      + $eb86d391,21);}

        Inc(CurrentHash[0],A);
        Inc(CurrentHash[1],B);
        Inc(CurrentHash[2],C);
        Inc(CurrentHash[3],D);
        Index:= 0;
        FillChar(HashBuffer,Sizeof(HashBuffer),0);
      end;
```

Este es el procedimiento “compress” que donde están incluidas las rondas que son donde se hacen las operaciones lógicas del mensaje, el mensaje en bloques de 16 palabras. En primer lugar, se tiene la definición de variables donde “Data” es un arreglo de 16 espacios que almacena datos del tipo longint. También se definen las variables “A”, “B”, “C”, “D”, del tipo longint, en el bucle del procedimiento en primer lugar se mueven los datos almacenado en “HashBuffer” en el tamaño del arreglo “Data”. Después se guardan los valores del arreglo “CurrenHash” que tiene 4 espacios donde están almacenado los valores iniciales del buffer MD en las variables “A”,” B”,” C”,” D” en orden alfabético desde el espacio 0 al 4.

Se llega a la parte de las rondas donde se ejecutan las operaciones lógicas un numero definido de veces según el largo de la entrada cifrar, después de estas series de operaciones el resultado se vuelve ilegible y se rellena la matriz de salida alfanumérica de un largo de 128 bits.

Listado 8-5 Función “LRot32”

```
Function LRot32(a, b: longint): longint ;
VAR
  LR_OUT : LONGINT  ;

Begin
   { Result:= (a shl b) or (a shr (32-b));}
  ASM
     MOV EAX,a
     MOV EBX,EAX
     MOV ECX,b
     SHL EAX,CL

     MOV DL,CL
     MOV CL,32
     SUB CL,DL
     SHR EBX,CL

     OR  EAX,EBX
     MOV LR_OUT,EAX
  END;
  LROT32 := LR_OUT;
End;
```

Esta función rota a la derecha lo almacenado en “a” en “b” bits sin signos.

Listado 8-6 Función “sum2”

```
Function sum2 (data1,data2:longint):longint  ;
Var  add_32_out :Longint ;
Begin
  ASM
     MOV EAX,data1
     MOV EBX,data2
     ADD EAX,EBX
     MOV ADD_32_OUT,EAX
  End;
  sum2 :=  add_32_out ;
End;
```

sum2 toma el valor almacenado en la variable “data1” y le suma “data2” sin signos ambas del tipo longint.

Listado 8-7 Función “sum4”

```
Function sum4 (data1,data2,data3,data4:Longint) :
longint ;
 Var  suma_4: longint ;
 Begin
  suma_4 := data1;
  suma_4 := sum2 (suma_4,data2);
  suma_4 := sum2 (suma_4,data3);
  suma_4 := sum2 (suma_4,data4);
  sum4   := suma_4
 End;
```

Esta función guarda el valor almacenado en ”data1” y lo guarda en “suma_4”, luego se llama a la función “sum2” que suma dos números sin signos para el primer caso se suma lo almacenado en “suma_4” con “dat2” y el resultado lo almacena en “suma_4” así suma todos los datos almacenado en “sum4” y el resultado final de la suma de las 4 variables lo almacena en “sum4”.

Listado 8-8 Procedimiento “OP1”

```
PROCEDURE OP1 (Var DAT0:LONGINT;
dat1,dat2,dat3,dat4,dat5,dat6,dat7,dat8,dat9:Longint);
 Var  suma_L: longint ;
 Begin
  suma_L := dat3 xor (dat4 and (dat5 xor dat6));
  suma_L := sum2 (suma_L,dat2);
  suma_L := sum2 (suma_L,dat7);
  suma_L := sum2 (suma_L,dat8);
  suma_L := LRot32(suma_L,dat9);
  suma_L := sum2 (suma_L,dat1);
  dat0   := suma_L
 End;
```

Este procedimiento es llamado en “compress” tomando los valores de las variables que se encuentran dentro de éste y almacenándolo en las variables definidas dentro de “OP1” se puede

ingresar al procedimiento donde se define la variable “suma_L”, que almacenara los resultados de las operaciones lógicas definidas en cada round, para “OP1” se ejecutarán todas las operaciones y el resultado final de “suma_L” será almacenado en “dat0” para este procedimiento se ejecutan las operaciones lógicas de dos round, es decir 32 palabras de 16 cada round , luego las siguientes operaciones serán ejecutadas por los procedimientos ”OP2” y “OP3”.

Listado 8-9 Procedimiento “OP2”

```
PROCEDURE OP2 (Var DAT0:LONGINT;
dat1,dat2,dat3,dat4,dat5,dat6,dat7,dat8:Longint);
 Var suma_L: longint ;
 Begin
  suma_L := dat3 xor (dat4 xor dat5);
  suma_L := sum2 (suma_L,dat2);
  suma_L := sum2 (suma_L,dat6);
  suma_L := sum2 (suma_L,dat7);
  suma_L := LRot32(suma_L,dat8);
  suma_L := sum2 (suma_L,dat1);
  dat0   := suma_L
 End;
```

En este procedimiento se operan los datos ingresados de manera lógicamente distintas a “OP1”, pero el fin es el mismo llevar a cabo uno de los round que es hacer 16 operaciones lógicas con los valores de “HashBuffer” que se movieron a la variable “data” y los del “CurrenHash” que se almacenaron en las variables “A”, “B”, “C”,”D”, con esto se cumple el procesado de los bloques de 16 palabras, que entrega una palabra de 32 bits por cada round. El resultado final al operar cada uno de los procedimientos lógicos se almacena en “dat0”.

Listado 8-10 Procedimiento “OP3”

```
PROCEDURE OP3 (Var DAT0:LONGINT;
dat1,dat2,dat3,dat4,dat5,dat6,dat7,dat8:Longint);
 Var  suma_L: longint ;
 Begin
  suma_L := dat3 xor (Dat4 or (not dat5));
  suma_L := sum2 (suma_L,dat2);
  suma_L := sum2 (suma_L,dat6);
  suma_L := sum2 (suma_L,dat7);
  suma_L := LRot32(suma_L,dat8);
  suma_L := sum2 (suma_L,dat1);
  dat0   := suma_L
 End;
```

Este procedimiento es el que ejecuta el último round de operaciones lógicas, se define la variable “suma_L” donde se almacenan los resultados provisorios de cada una de las operaciones, así se puede tomar el valor en la siguiente operación, y el resultado final de las operaciones del procedimiento es almacenado en la variable “dat0”.

Al finalizar el procedimiento “OP3” se vuelve a analizar “Compress” para finalizar con las últimas instrucciones dentro de este procedimiento la primera de esta serie de instrucciones después de ejecutado “OP3” es incrementar el valor de los datos almacenado en el arreglo “CurrentHash” que posee cuatro espacios que almacenan variables operadas lógicamente y se incrementa en el valor de las variables “A”, “B”,”C”,” D”, esta últimas incrementarán los valores de “CurrentHash” en orden alfabético. Por lo cual el resultado final son 16 palabras de un byte. Por último, se rellenan los espacios de la variable “HashBuffer” con ceros y le da el valor cero a la variable “index” y termina el procedimiento “Compress”.

Listado 8-11 Procedimiento “final”

```
Procedure Final(var Digest);
begin
  HashBuffer[Index]:= $80;
  if Index>= 56 then Compress;
  mem  [ofs (HashBuffer[56])] := LenLo;//
  mem  [ofs (HashBuffer[60])] := LenHi;
  Compress;
  Move(CurrentHash,Digest,Sizeof(CurrentHash));
  Burn;
end;
```

Este procedimiento se llamará ingresando un valor en “a” que es un array de 16 espacio del tipo byte, dentro de las acciones a ejecutar esta darle el valor “$80” a la posición de valor “index” de “HashBuffer” en este caso se almacenará en el primer espacio. Luego hay una condición “if ” donde si “index” es mayor o igual a 56 entonces se ejecutara “Compress” que es un procedimiento definido anteriormente, el siguiente paso a ejecutar es “mem [ofs (HashBuffer[56])]” que llena desde el espacio 56 el arreglo con el valor de “LenLo”, y para “mem [ofs (HashBuffer[60])]” se llenará desde el espacio 60 con el valor de “LenHi”, para asi ejecutar “Compress”, la siguiente acción a ejecutar es mover los valores de “CurrentHash” obtenidos después de haber ejecutado “Compress” y moverlos a “Digest” que se llena con el tamaño de “CurrentHash” que guarda un longint en 4 byte de “Digest”, luego de ésto se ejecuta el procedimiento “burn” definido anteriormente que devuelve a cero las variables y los arrays antes rellenados.

8.3 Ciclo principal

Listado 8-11 Procedimiento “final”

```
procedure Final(var Digest);
begin
  HashBuffer[Index]:= $80;
  if Index>= 56 then Compress;
  mem  [ofs (HashBuffer[56])] := LenLo;//
  mem  [ofs (HashBuffer[60])] := LenHi;
  Compress;
  Move(CurrentHash,Digest,Sizeof(CurrentHash));
  Burn;
end;
```

Este procedimiento se llamará ingresando un valor en “a” que es un array de 16 espacio del tipo byte, dentro de las acciones a ejecutar esta darle el valor “$80” a la posición de valor “index” de “HashBuffer” en este caso se almacenara en el primer espacio. Luego hay una condición “if ” donde si “index” es mayor o igual a 56 entonces se ejecutará “Compress” que es un procedimiento definido anteriormente, el siguiente paso a establecer es “mem [ofs (HashBuffer[56])]” que llena desde el espacio 56 el arreglo con el valor de “LenLo”, y para “mem [ofs (HashBuffer[60])]” se llenara desde el espacio 60 con el valor de “LenHi”, para así efectuar “Compress”, la siguiente acción a ejecutar es mover los valores de “CurrentHash” obtenidos después de haber ejecutado “Compress” y moverlos a “Digest” que se llena con el tamaño de “CurrentHash” que guarda un longint en 4 byte de “Digest”, luego de esto se realiza el procedimiento “burn” definido anteriormente que devuelve a cero las variables y se te termina el procedimiento “Final”.

8.4 Compilacion

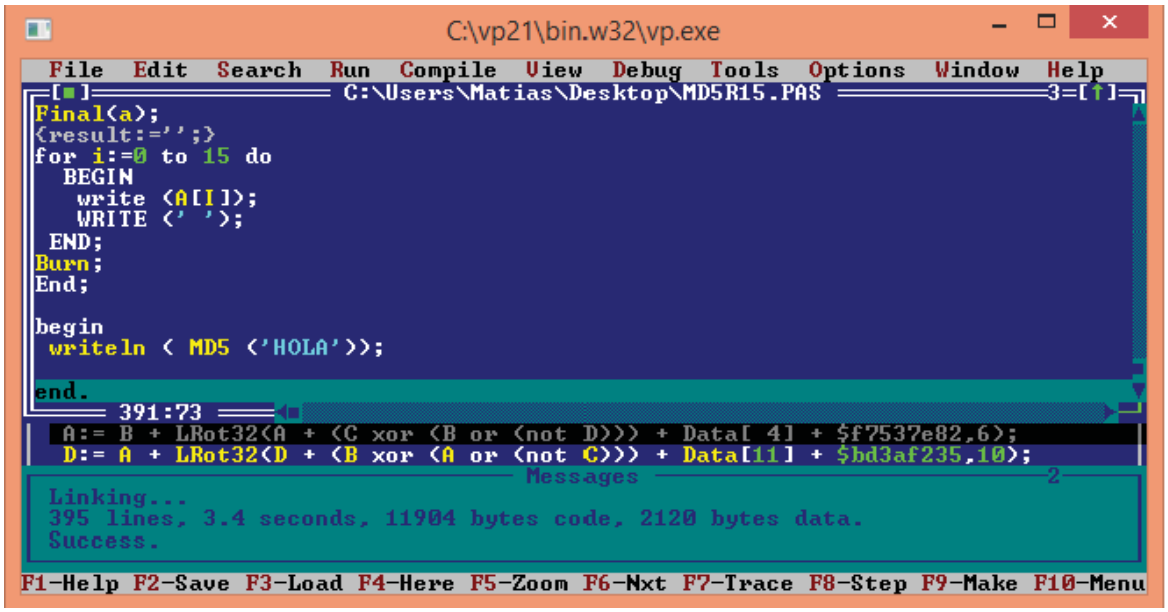


Ilustración 24 compilador del programa semilla "HOLA"

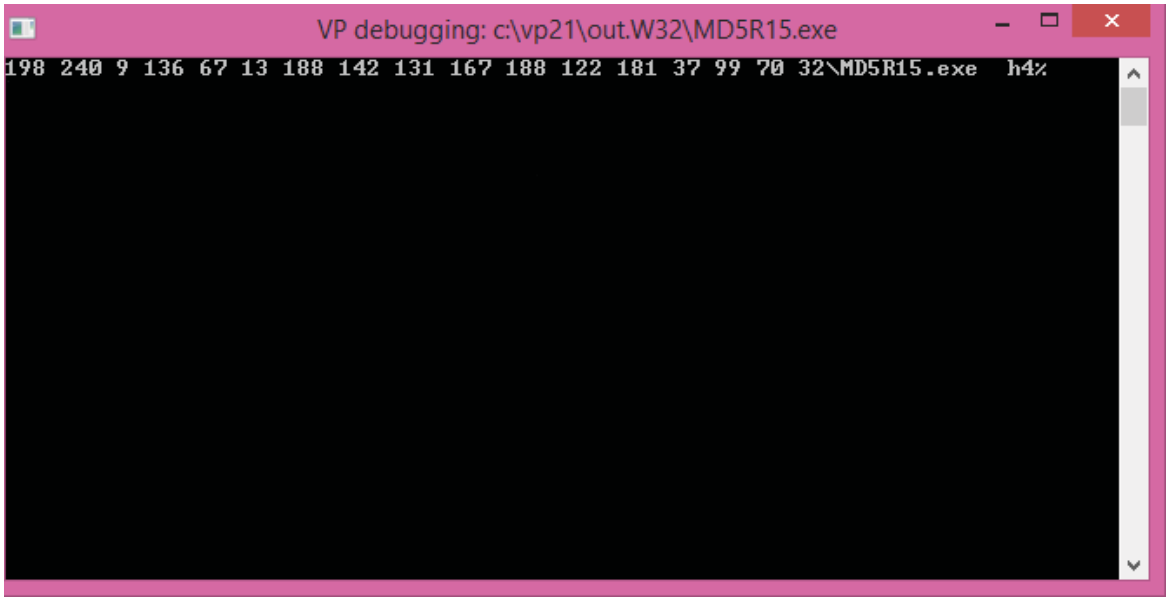


Ilustración 25 Resumen semilla "HOLA"

En la ilustración 2 se muestra el resumen del Hash para una semilla arbitraria en este caso “Hola”, la cual posee 128 bits que es entregada en decimal.

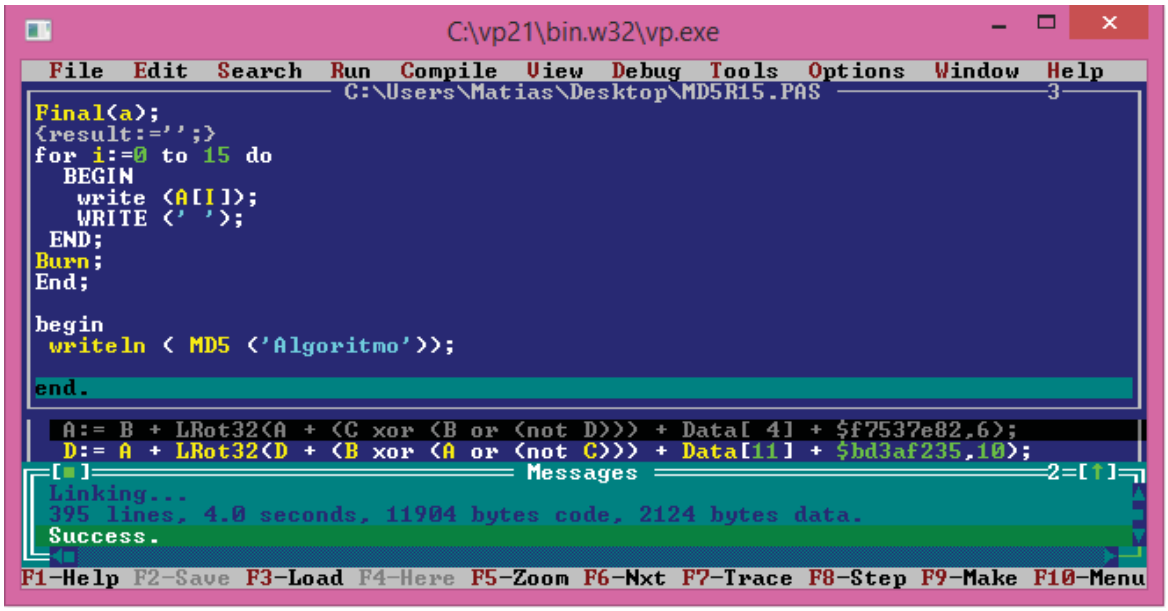


Ilustración 26 Compilador programa semilla "Algoritmo"

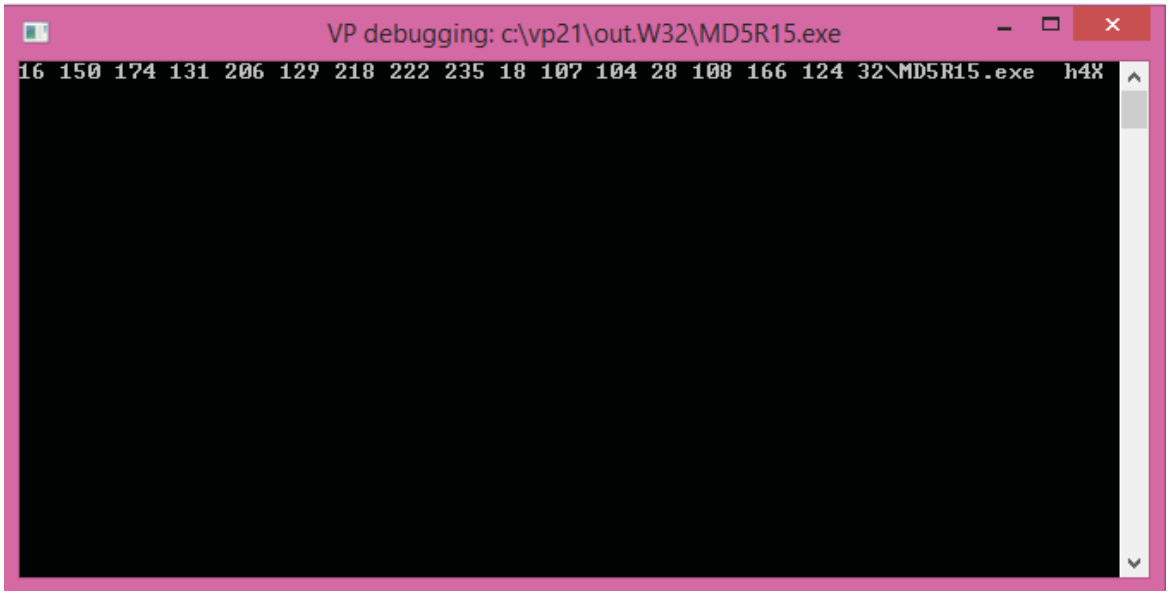


Ilustración 27 Resumen semilla "Algoritmo"

En la ilustración 28 se muestra el resumen del HASH para una semilla arbitraria en este caso “Algoritmo”, la cual posee 128 bits que es entregada en decimal.

9 Transferencia de datos con MD5

Hasta el momento solo se habían encriptados semillas arbitrarias a través del algoritmo MD5, lo cual entrega un HASH de 128bits (16 byte) de largo, lo cual permite la transmisión de archivos en tiempo real.

Generación de tabla con valores hexadecimal

Listado 9-1 Tabla con valores de HASH de todos los hexadecimales

```
Const enig  : Array [0..$FF] OF Array [0..$F] of Byte =


      (

($93,$B8,$85,$AD,$FE,$0D,$A0,$89,$CD,$F6,$34,$90,$4F,$D5,$9F,$71),    {
$00}

($55,$A5,$40,$08,$AD,$1B,$A5,$89,$AA,$21,$0D,$26,$29,$C1,$DF,$41),    {
$01}

($9E,$68,$8C,$58,$A5,$48,$7B,$8E,$AF,$69,$C9,$E1,$00,$5A,$D0,$BF),    {
$02}

($86,$66,$68,$35,$06,$AA,$CD,$90,$0B,$BD,$5A,$74,$AC,$4E,$DF,$68),    {
$03}

($EC,$7F,$7E,$7B,$B4,$37,$42,$CE,$86,$81,$45,$F7,$1D,$37,$B5,$3C),    {
$04}

($8B,$B6,$C1,$78,$38,$64,$3F,$96,$91,$CC,$6A,$4D,$E6,$C5,$17,$09),    {
$05}

($06,$EC,$A1,$B4,$37,$C7,$90,$4C,$C3,$CE,$65,$46,$C8,$11,$01,$10),    {
$06}

($89,$E7,$4E,$64,$0B,$8C,$46,$25,$7A,$29,$DE,$06,$16,$79,$4D,$5D),    {
$07}

($E2,$BA,$90,$5B,$F3,$06,$F4,$6F,$AC,$A2,$23,$D3,$CB,$20,$E2,$CF),    {
$08}

($5E,$73,$2A,$18,$78,$BE,$23,$42,$DB,$FE,$FF,$5F,$E3,$CA,$5A,$A3),    {
$09}

($68,$B3,$29,$DA,$98,$93,$E3,$40,$99,$C7,$D8,$AD,$5C,$B9,$C9,$40),    {
$0A}

($13,$C8,$FF,$D9,$77,$01,$37,$03,$A7,$01,$CF,$8E,$11,$DE,$AC,$65),    {
$0B}
```

```
($58,$C8,$95,$62,$F5,$8F,$D2,$76,$F5,$92,$42,$00,$68,$DB,$8C,$09),    {
$0C}

($DC,$B9,$BE,$2F,$60,$4E,$5D,$F9,$1D,$EB,$96,$59,$BE,$D4,$74,$8D),    {
$0D}

($4D,$ED,$B2,$24,$0A,$1E,$0F,$03,$8D,$CD,$C8,$B3,$DE,$92,$26,$4C),    {
$0E}

($D8,$38,$69,$1E,$5D,$4A,$D0,$68,$79,$CA,$72,$14,$42,$E8,$83,$D4),    {
$0F}

($6B,$31,$BD,$FA,$7F,$9B,$FE,$CE,$26,$33,$81,$FF,$A9,$1B,$D6,$A9),    {
$10}

($47,$ED,$73,$3B,$8D,$10,$BE,$22,$5E,$CE,$BA,$34,$4D,$53,$35,$86),    {
$11}

($A8,$44,$56,$19,$AB,$D0,$8F,$3B,$A0,$EB,$FC,$B3,$11,$83,$F7,$F9),    {
$12}

($FF,$E5,$1D,$3E,$7D,$82,$97,$23,$75,$88,$70,$4E,$ED,$DC,$6A,$B2),    {
$13}

($15,$F4,$1A,$2E,$96,$BA,$E3,$41,$DD,$E4,$85,$BB,$0E,$78,$F4,$85),    {
$14}

($F5,$A7,$E4,$77,$CD,$30,$42,$B4,$9A,$90,$85,$D6,$23,$07,$CD,$28),    {
$15}

($BF,$6D,$6C,$81,$9E,$C9,$75,$B0,$43,$AE,$C5,$02,$16,$7C,$3D,$15),    {
$16}

($84,$FF,$14,$FA,$45,$BE,$3C,$A4,$73,$9E,$7C,$02,$77,$17,$A5,$41),    {
$17}

($CB,$A8,$1A,$CD,$53,$FD,$77,$19,$F0,$AA,$94,$95,$93,$5A,$87,$2B),    {
$18}

($E5,$EA,$7F,$B5,$1F,$F2,$7A,$20,$C3,$F6,$22,$DF,$66,$B9,$AC,$DC),    {
$19}

($BE,$BE,$43,$A1,$3D,$63,$20,$B4,$C6,$75,$19,$58,$BF,$53,$98,$A7),    {
$1A}

($F6,$16,$C8,$3F,$2F,$0F,$18,$82,$65,$C7,$00,$4D,$81,$D4,$57,$23),    {
$1B}

($03,$98,$B4,$09,$0F,$24,$AD,$BC,$CC,$21,$82,$19,$F5,$74,$6B,$10),    {
$1C}

($EB,$25,$9E,$DB,$AA,$60,$8E,$B2,$20,$80,$46,$61,$9B,$48,$46,$68),    {
$1D}

($7B,$C7,$2A,$07,$67,$D2,$37,$BE,$4D,$A3,$0A,$CE,$19,$1A,$CD,$C2),    {
$1E}

($AD,$1E,$41,$CE,$BD,$43,$E6,$4A,$F1,$A2,$8D,$4D,$70,$DC,$9E,$30),    {
$1F}

($72,$15,$EE,$9C,$7D,$9D,$C2,$29,$D2,$92,$1A,$40,$E8,$99,$EC,$5F),    {
$20}

($90,$33,$E0,$E3,$05,$F2,$47,$C0,$C3,$C8,$0D,$0C,$78,$48,$C8,$B3),    {
$21}

($B1,$58,$35,$F1,$33,$FF,$2E,$27,$C7,$CB,$28,$11,$7B,$FA,$E8,$F4),    {
$22}
```

```
($01,$AB,$FC,$75,$0A,$0C,$94,$21,$67,$65,$1C,$40,$D0,$88,$53,$1D),    {
$23}

($C3,$E9,$7D,$D6,$E9,$7F,$B5,$12,$56,$88,$C9,$7F,$36,$72,$0C,$BE),    {
$24}

($0B,$CE,$F9,$C4,$5B,$D8,$A4,$8E,$DA,$1B,$26,$EB,$0C,$61,$C8,$69),    {
$25}

($6C,$FF,$04,$78,$54,$F1,$9A,$C2,$AA,$52,$AA,$C5,$1B,$F3,$AF,$4A),    {
$26}

($35,$90,$CB,$8A,$F0,$BB,$B9,$E7,$8C,$34,$3B,$52,$B9,$37,$73,$C9),    {
$27}

($84,$C4,$04,$73,$41,$4C,$AF,$2E,$D4,$A7,$B1,$28,$3E,$48,$BB,$F4),    {
$28}

($93,$71,$D7,$A2,$E3,$AE,$86,$A0,$0A,$AB,$47,$71,$E3,$9D,$25,$5D),    {
$29}

($33,$89,$DA,$E3,$61,$AF,$79,$B0,$4C,$9C,$8E,$70,$57,$F6,$0C,$C6),    {
$2A}

($26,$B1,$72,$25,$B6,$26,$FB,$92,$38,$84,$9F,$D6,$0E,$AB,$DF,$60),    {
$2B}

($C0,$CB,$5F,$0F,$CF,$23,$9A,$B3,$D9,$C1,$FC,$D3,$1F,$FF,$1E,$FC),    {
$2C}

($33,$6D,$5E,$BC,$54,$36,$53,$4E,$61,$D1,$6E,$63,$DD,$FC,$A3,$27),    {
$2D}

($50,$58,$F1,$AF,$83,$88,$63,$3F,$60,$9C,$AD,$B7,$5A,$75,$DC,$9D),    {
$2E}

($66,$66,$CD,$76,$F9,$69,$56,$46,$9E,$7B,$E3,$9D,$75,$0C,$C7,$D9),    {
$2F}

($CF,$CD,$20,$84,$95,$D5,$65,$EF,$66,$E7,$DF,$F9,$F9,$87,$64,$DA),    {
$30}

($C4,$CA,$42,$38,$A0,$B9,$23,$82,$0D,$CC,$50,$9A,$6F,$75,$84,$9B),    {
$31}

($C8,$1E,$72,$8D,$9D,$4C,$2F,$63,$6F,$06,$7F,$89,$CC,$14,$86,$2C),    {
$32}

($EC,$CB,$C8,$7E,$4B,$5C,$E2,$FE,$28,$30,$8F,$D9,$F2,$A7,$BA,$F3),    {
$33}

($A8,$7F,$F6,$79,$A2,$F3,$E7,$1D,$91,$81,$A6,$7B,$75,$42,$12,$2C),    {
$34}

($E4,$DA,$3B,$7F,$BB,$CE,$23,$45,$D7,$77,$2B,$06,$74,$A3,$18,$D5),    {
$35}

($16,$79,$09,$1C,$5A,$88,$0F,$AF,$6F,$B5,$E6,$08,$7E,$B1,$B2,$DC),    {
$36}

($8F,$14,$E4,$5F,$CE,$EA,$16,$7A,$5A,$36,$DE,$DD,$4B,$EA,$25,$43),    {
$37}

($C9,$F0,$F8,$95,$FB,$98,$AB,$91,$59,$F5,$1F,$D0,$29,$7E,$23,$6D),    {
$38}

($45,$C4,$8C,$CE,$2E,$2D,$7F,$BD,$EA,$1A,$FC,$51,$C7,$C6,$AD,$26),    {
$39}
```

```
($85,$3A,$E9,$0F,$03,$51,$32,$4B,$D7,$3E,$A6,$15,$E6,$48,$75,$17),    {
$3A}

($9E,$EC,$B7,$DB,$59,$D1,$6C,$80,$41,$7C,$72,$D1,$E1,$F4,$FB,$F1),    {
$3B}

($52,$4A,$50,$78,$21,$78,$99,$80,$21,$A8,$8B,$8C,$D4,$C8,$DC,$D8),    {
$3C}

($43,$EC,$3E,$5D,$EE,$6E,$70,$6A,$F7,$76,$6F,$FF,$EA,$51,$27,$21),    {
$3D}

($CE,$DF,$8D,$A0,$54,$66,$BB,$54,$70,$82,$68,$B3,$C6,$94,$A7,$8F),    {
$3E}

($D1,$45,$7B,$72,$C3,$FB,$32,$3A,$26,$71,$12,$5A,$EF,$3E,$AB,$5D),    {
$3F}

($51,$8E,$D2,$95,$25,$73,$8C,$EB,$DA,$C4,$9C,$49,$E6,$0E,$A9,$D3),    {
$40}

($7F,$C5,$62,$70,$E7,$A7,$0F,$A8,$1A,$59,$35,$B7,$2E,$AC,$BE,$29),    {
$41}

($9D,$5E,$D6,$78,$FE,$57,$BC,$CA,$61,$01,$40,$95,$7A,$FA,$B5,$71),    {
$42}

($0D,$61,$F8,$37,$0C,$AD,$1D,$41,$2F,$80,$B8,$4D,$14,$3E,$12,$57),    {
$43}

($F6,$23,$E7,$5A,$F3,$0E,$62,$BB,$D7,$3D,$6D,$F5,$B5,$0B,$B7,$B5),    {
$44}

($3A,$3E,$A0,$0C,$FC,$35,$33,$2C,$ED,$F6,$E5,$E9,$A3,$2E,$94,$DA),    {
$45}

($80,$06,$18,$94,$30,$25,$31,$5F,$86,$9E,$4E,$1F,$09,$47,$10,$12),    {
$46}

($DF,$CF,$28,$D0,$73,$45,$69,$A6,$A6,$93,$BC,$81,$94,$DE,$62,$BF),    {
$47}

($C1,$D9,$F5,$0F,$86,$82,$5A,$1A,$23,$02,$EC,$24,$49,$C1,$71,$96),    {
$48}

($DD,$75,$36,$79,$4B,$63,$BF,$90,$EC,$CF,$D3,$7F,$9B,$14,$7D,$7F),    {
$49}

($FF,$44,$57,$0A,$CA,$82,$41,$91,$48,$70,$AF,$BC,$31,$0C,$DB,$85),    {
$4A}

($A5,$F3,$C6,$A1,$1B,$03,$83,$9D,$46,$AF,$9F,$B4,$3C,$97,$C1,$88),    {
$4B}

($D2,$0C,$AE,$C3,$B4,$8A,$1E,$EF,$16,$4C,$B4,$CA,$81,$BA,$25,$87),    {
$4C}

($69,$69,$1C,$7B,$DC,$C3,$CE,$6D,$5D,$8A,$13,$61,$F2,$2D,$04,$AC),    {
$4D}

($8D,$9C,$30,$7C,$B7,$F3,$C4,$A3,$28,$22,$A5,$19,$22,$D1,$CE,$AA),    {
$4E}

($F1,$86,$21,$77,$53,$C3,$7B,$9B,$9F,$95,$8D,$90,$62,$08,$50,$6E),    {
$4F}

($44,$C2,$9E,$DB,$10,$3A,$28,$72,$F5,$19,$AD,$0C,$9A,$0F,$DA,$AA),    {
$50}
```

```
($F0,$95,$64,$C9,$CA,$56,$85,$0D,$4C,$D6,$B3,$31,$9E,$54,$1A,$EE),    {
$51}

($E1,$E1,$D3,$D4,$05,$73,$12,$7E,$9E,$E0,$48,$0C,$AF,$12,$83,$D6),    {
$52}

($5D,$BC,$98,$DC,$C9,$83,$A7,$07,$28,$BD,$08,$2D,$1A,$47,$54,$6E),    {
$53}

($B9,$EC,$E1,$8C,$95,$0A,$FB,$FA,$6B,$0F,$DB,$FA,$4F,$F7,$31,$D3),    {
$54}

($4C,$61,$43,$60,$DA,$93,$C0,$A0,$41,$B2,$2E,$53,$7D,$E1,$51,$EB),    {
$55}

($52,$06,$56,$0A,$30,$6A,$2E,$08,$5A,$43,$7F,$D2,$58,$EB,$57,$CE),    {
$56}

($61,$E9,$C0,$6E,$A9,$A8,$5A,$50,$88,$A4,$99,$DF,$64,$58,$D2,$76),    {
$57}

($02,$12,$9B,$B8,$61,$06,$1D,$1A,$05,$2C,$59,$2E,$2D,$C6,$B3,$83),    {
$58}

($57,$CE,$C4,$13,$7B,$61,$4C,$87,$CB,$4E,$24,$A3,$D0,$03,$A3,$E0),    {
$59}

($21,$C2,$E5,$95,$31,$C8,$71,$01,$56,$D3,$4A,$3C,$30,$AC,$81,$D5),    {
$5A}

($81,$54,$17,$26,$7F,$76,$F6,$F4,$60,$A4,$A6,$1F,$9D,$B7,$5F,$DB),    {
$5B}

($28,$D3,$97,$E8,$73,$06,$B8,$63,$1F,$3E,$D8,$0D,$85,$8D,$35,$F0),    {
$5C}

($0F,$BD,$17,$76,$E1,$AD,$22,$C5,$9A,$70,$80,$D3,$5C,$7F,$D4,$DB),    {
$5D}

($7E,$6A,$2A,$FE,$55,$1E,$06,$7A,$75,$FA,$FA,$CF,$47,$A6,$D9,$81),    {
$5E}

($B1,$4A,$7B,$80,$59,$D9,$C0,$55,$95,$4C,$92,$67,$4C,$E6,$00,$32),    {
$5F}

($83,$33,$44,$D5,$E1,$43,$2D,$A8,$2E,$F0,$2E,$13,$01,$47,$7C,$E8),    {
$60}

($0C,$C1,$75,$B9,$C0,$F1,$B6,$A8,$31,$C3,$99,$E2,$69,$77,$26,$61),    {
$61}

($92,$EB,$5F,$FE,$E6,$AE,$2F,$EC,$3A,$D7,$1C,$77,$75,$31,$57,$8F),    {
$62}

($4A,$8A,$08,$F0,$9D,$37,$B7,$37,$95,$64,$90,$38,$40,$8B,$5F,$33),    {
$63}

($82,$77,$E0,$91,$0D,$75,$01,$95,$B4,$48,$79,$76,$16,$E0,$91,$AD),    {
$64}

($E1,$67,$17,$97,$C5,$2E,$15,$F7,$63,$38,$0B,$45,$E8,$41,$EC,$32),    {
$65}

($8F,$A1,$4C,$DD,$75,$4F,$91,$CC,$65,$54,$C9,$E7,$19,$29,$CC,$E7),    {
$66}

($B2,$F5,$FF,$47,$43,$66,$71,$B6,$E5,$33,$D8,$DC,$36,$14,$84,$5D),    {
$67}
```

```
($25,$10,$C3,$90,$11,$C5,$BE,$70,$41,$82,$42,$3E,$3A,$69,$5E,$91),    {
$68}

($86,$5C,$0C,$0B,$4A,$B0,$E0,$63,$E5,$CA,$A3,$38,$7C,$1A,$87,$41),    {
$69}

($36,$3B,$12,$2C,$52,$8F,$54,$DF,$4A,$04,$46,$B6,$BA,$B0,$55,$15),    {
$6A}

($8C,$E4,$B1,$6B,$22,$B5,$88,$94,$AA,$86,$C4,$21,$E8,$75,$9D,$F3),    {
$6B}

($2D,$B9,$5E,$8E,$1A,$92,$67,$B7,$A1,$18,$85,$56,$B2,$01,$3B,$33),    {
$6C}

($6F,$8F,$57,$71,$50,$90,$DA,$26,$32,$45,$39,$88,$D9,$A1,$50,$1B),    {
$6D}

($7B,$8B,$96,$5A,$D4,$BC,$A0,$E4,$1A,$B5,$1D,$E7,$B3,$13,$63,$A1),    {
$6E}

($D9,$56,$79,$75,$21,$34,$A2,$D9,$EB,$61,$DB,$D7,$B9,$1C,$4B,$CC),    {
$6F}

($83,$87,$8C,$91,$17,$13,$38,$90,$2E,$0F,$E0,$FB,$97,$A8,$C4,$7A),    {
$70}

($76,$94,$F4,$A6,$63,$16,$E5,$3C,$8C,$DD,$9D,$99,$54,$BD,$61,$1D),    {
$71}

($4B,$43,$B0,$AE,$E3,$56,$24,$CD,$95,$B9,$10,$18,$9B,$3D,$C2,$31),    {
$72}

($03,$C7,$C0,$AC,$E3,$95,$D8,$01,$82,$DB,$07,$AE,$2C,$30,$F0,$34),    {
$73}

($E3,$58,$EF,$A4,$89,$F5,$80,$62,$F1,$0D,$D7,$31,$6B,$65,$64,$9E),    {
$74}

($7B,$77,$4E,$FF,$E4,$A3,$49,$C6,$DD,$82,$AD,$4F,$4F,$21,$D3,$4C),    {
$75}

($9E,$36,$69,$D1,$9B,$67,$5B,$D5,$70,$58,$FD,$46,$64,$20,$5D,$2A),    {
$76}

($F1,$29,$01,$86,$A5,$D0,$B1,$CE,$AB,$27,$F4,$E7,$7C,$0C,$5D,$68),    {
$77}

($9D,$D4,$E4,$61,$26,$8C,$80,$34,$F5,$C8,$56,$4E,$15,$5C,$67,$A6),    {
$78}

($41,$52,$90,$76,$95,$94,$46,$0E,$2E,$48,$59,$22,$90,$4F,$34,$5D),    {
$79}

($FB,$AD,$E9,$E3,$6A,$3F,$36,$D3,$D6,$76,$C1,$B8,$08,$45,$1D,$D7),    {
$7A}

($F9,$5B,$70,$FD,$C3,$08,$85,$60,$73,$2A,$5A,$C1,$35,$64,$45,$06),    {
$7B}

($B9,$98,$34,$BC,$19,$BB,$AD,$24,$58,$0B,$3A,$DF,$A0,$4F,$B9,$47),    {
$7C}

($CB,$B1,$84,$DD,$8E,$05,$C9,$70,$9E,$5D,$CA,$ED,$AA,$04,$95,$CF),    {
$7D}

($4C,$76,$1F,$17,$0E,$01,$68,$36,$FF,$84,$49,$82,$02,$B9,$98,$27),    {
$7E}
```

```
($83,$AC,$B6,$E6,$7E,$50,$E3,$1D,$B6,$ED,$34,$1D,$D2,$DE,$15,$95),    {
$7F}

($8D,$39,$DD,$7E,$EF,$11,$5E,$A6,$97,$54,$46,$EF,$40,$82,$95,$1F),    {
$80}

($CD,$25,$04,$1F,$9F,$36,$81,$1B,$04,$AB,$30,$15,$80,$5F,$E8,$16),    {
$81}

($59,$2E,$EC,$8D,$32,$EB,$49,$0E,$69,$19,$25,$3D,$9C,$A6,$37,$4D),    {
$82}

($05,$D8,$58,$04,$DD,$3E,$68,$9E,$1F,$1A,$0A,$AA,$19,$75,$FB,$4C),    {
$83}

($EC,$63,$1D,$73,$35,$AB,$EC,$D3,$18,$F0,$9F,$56,$51,$5E,$D6,$3C),    {
$84}

($A0,$39,$20,$E5,$99,$42,$02,$F7,$7B,$9C,$71,$39,$41,$B5,$E0,$55),    {
$85}

($DC,$AB,$A5,$D0,$E9,$EA,$E2,$78,$73,$A5,$1C,$B5,$01,$69,$A4,$6A),    {
$86}

($8F,$EC,$37,$87,$60,$F1,$E7,$E7,$08,$E9,$F5,$7E,$DA,$FC,$28,$FC),    {
$87}

($76,$44,$63,$46,$14,$7D,$6A,$63,$18,$0D,$28,$E0,$E6,$A6,$B0,$72),    {
$88}

($28,$54,$27,$2F,$EC,$04,$4D,$0B,$DB,$16,$DE,$12,$CB,$62,$D0,$7E),    {
$89}

($1D,$94,$85,$37,$44,$51,$32,$EB,$F7,$46,$BD,$33,$D5,$49,$4B,$70),    {
$8A}

($34,$8D,$D9,$E9,$74,$84,$8A,$61,$E8,$AB,$E2,$DF,$FC,$FC,$CE,$93),    {
$8B}

($97,$77,$5B,$A0,$9E,$AC,$63,$B5,$DC,$C9,$8A,$94,$B2,$F7,$79,$B9),    {
$8C}

($CF,$87,$DE,$09,$91,$40,$66,$FD,$E4,$6E,$4A,$24,$F2,$2E,$A5,$35),    {
$8D}

($F1,$66,$3A,$BA,$9F,$FA,$E5,$33,$8B,$63,$82,$A2,$4B,$2E,$53,$77),    {
$8E}

($32,$B4,$01,$96,$01,$54,$E8,$6D,$27,$BA,$06,$C6,$B9,$D3,$40,$8C),    {
$8F}

($BC,$9A,$BF,$1B,$E5,$9F,$FC,$E1,$80,$25,$1D,$4C,$CE,$75,$5F,$E2),    {
$90}

($40,$41,$22,$02,$89,$23,$18,$D5,$28,$22,$E6,$21,$95,$A6,$57,$62),    {
$91}

($68,$5D,$59,$0A,$89,$F3,$52,$D1,$06,$44,$DD,$98,$5D,$A2,$A9,$A1),    {
$92}

($5C,$5A,$A2,$BA,$6E,$48,$A0,$E5,$59,$C1,$49,$CD,$D4,$CE,$7A,$9E),    {
$93}

($9D,$A8,$AA,$9F,$77,$11,$9C,$92,$19,$B1,$03,$92,$2A,$74,$E9,$1C),    {
$94}

($75,$8A,$A1,$74,$1F,$4B,$2F,$D2,$E8,$6C,$F8,$F2,$D5,$21,$D5,$A5),    {
$95}
```

```
($3C,$C7,$37,$46,$E4,$10,$9C,$72,$7B,$91,$0A,$9D,$7E,$39,$F0,$27),    {
$96}

($C4,$44,$B5,$80,$07,$9E,$FB,$1F,$E4,$08,$F1,$7F,$02,$9E,$5D,$35),    {
$97}

($47,$16,$B0,$7C,$E6,$84,$75,$4B,$6C,$B0,$22,$3C,$8A,$F4,$29,$B1),    {
$98}

($6D,$4A,$60,$F6,$F3,$5F,$B2,$CE,$DA,$4D,$F4,$FE,$58,$B2,$C9,$BE),    {
$99}

($6D,$3A,$9B,$AE,$17,$22,$68,$50,$31,$07,$6A,$B0,$30,$94,$25,$FC),    {
$9A}

($B6,$3F,$46,$88,$1E,$5F,$CE,$A4,$13,$13,$27,$B2,$0B,$21,$80,$FD),    {
$9B}

($73,$D5,$9A,$7D,$52,$D3,$29,$EA,$FC,$90,$25,$E3,$AC,$8C,$24,$97),    {
$9C}

($0A,$47,$6D,$83,$EF,$9C,$EF,$4B,$CE,$7F,$90,$25,$52,$2B,$E3,$B5),    {
$9D}

($CE,$F5,$C0,$B7,$CE,$D3,$EE,$C9,$27,$31,$BC,$66,$E4,$ED,$0B,$9F),    {
$9E}

($DC,$5E,$CC,$DC,$F2,$93,$DB,$4C,$FA,$E5,$9A,$97,$C2,$8E,$75,$96),    {
$9F}

($9A,$F7,$C1,$17,$D9,$DE,$9A,$06,$FB,$A7,$A5,$F1,$EA,$5F,$CC,$2D),    {
$A0}

($73,$87,$F8,$D4,$47,$A0,$0C,$FD,$54,$8C,$F2,$F7,$57,$38,$A6,$60),    {
$A1}

($4F,$DF,$D7,$2B,$8D,$42,$29,$FF,$DD,$74,$27,$BF,$5B,$49,$00,$E6),    {
$A2}

($D5,$27,$CA,$07,$4D,$41,$2D,$9D,$0F,$FC,$84,$48,$72,$C4,$60,$3C),    {
$A3}

($F3,$7C,$6F,$38,$96,$B2,$C8,$5F,$BB,$D0,$1A,$E3,$2E,$47,$B4,$3F),    {
$A4}

($AB,$3A,$F8,$56,$6D,$DD,$20,$D7,$EF,$C9,$B3,$14,$AB,$E9,$07,$55),    {
$A5}

($60,$67,$A1,$76,$E5,$ED,$08,$F3,$7F,$90,$53,$7B,$9D,$BE,$76,$A5),    {
$A6}

($6B,$2B,$98,$FE,$A1,$1E,$51,$AF,$30,$43,$B1,$92,$F7,$19,$BD,$69),    {
$A7}

($85,$A4,$6A,$5D,$E1,$7B,$19,$AC,$7E,$B2,$9C,$AC,$CE,$9C,$BB,$04),    {
$A8}

($A2,$52,$C2,$C8,$5A,$9E,$77,$56,$D5,$BA,$5D,$A9,$94,$9D,$57,$ED),    {
$A9}

($9F,$E0,$F7,$24,$4A,$7D,$A1,$D3,$F5,$B3,$D2,$1F,$9B,$1E,$1E,$A8),    {
$AA}

($24,$08,$AD,$11,$F9,$EB,$83,$0D,$A7,$49,$E2,$A3,$6A,$29,$EF,$F7),    {
$AB}

($20,$A7,$F3,$0F,$B9,$F8,$E1,$45,$42,$2B,$66,$B3,$D4,$F4,$DA,$70),    {
$AC}
```

```
($3B,$EB,$9C,$F0,$EA,$B8,$CB,$F2,$21,$59,$90,$B4,$A6,$BD,$C2,$71),    {
$AD}

($02,$CB,$35,$22,$B3,$5E,$58,$09,$7E,$5F,$C3,$E9,$E0,$93,$D9,$B6),    {
$AE}

($00,$D9,$71,$2E,$C5,$EB,$70,$80,$7A,$73,$B8,$D2,$D6,$EA,$D9,$0D),    {
$AF}

($EC,$65,$5B,$6D,$A8,$B9,$26,$4A,$7C,$7C,$5E,$1A,$70,$64,$2F,$A7),    {
$B0}

($C6,$68,$53,$4D,$22,$0B,$AF,$21,$CA,$3C,$C6,$DF,$5B,$7E,$D1,$D5),    {
$B1}

($99,$41,$AE,$79,$62,$F3,$83,$0B,$2E,$90,$1D,$D6,$2C,$0A,$0F,$C9),    {
$B2}

($EC,$87,$21,$09,$73,$D5,$7B,$58,$29,$45,$CA,$FC,$5F,$D5,$DD,$63),    {
$B3}

($13,$73,$3A,$6E,$73,$C7,$D0,$23,$86,$51,$F7,$50,$7C,$47,$95,$53),    {
$B4}

($50,$F0,$35,$1E,$9C,$31,$BB,$1C,$8E,$18,$8E,$15,$2B,$95,$8F,$63),    {
$B5}

($DA,$63,$0C,$00,$D0,$4B,$E7,$F2,$4C,$DA,$AA,$CD,$01,$CF,$2E,$30),    {
$B6}

($DA,$E6,$65,$A6,$70,$92,$79,$61,$52,$B2,$9A,$AC,$43,$27,$FE,$44),    {
$B7}

($6A,$D3,$05,$F0,$9B,$A9,$2D,$FC,$5C,$CF,$32,$F2,$CE,$93,$B4,$60),    {
$B8}

($F3,$61,$E2,$57,$76,$07,$77,$89,$E0,$DB,$8C,$A9,$85,$BF,$36,$C5),    {
$B9}

($6B,$C1,$AF,$25,$84,$75,$EC,$12,$6D,$1B,$E6,$6E,$6B,$EE,$87,$3A),    {
$BA}

($D6,$84,$42,$A8,$D2,$CA,$DE,$05,$23,$24,$FC,$2A,$A5,$D7,$03,$9C),    {
$BB}

($4D,$CD,$E3,$76,$FB,$C2,$12,$F7,$3C,$0B,$00,$B7,$90,$9F,$C4,$CF),    {
$BC}

($AB,$AE,$57,$CB,$56,$2E,$CF,$29,$5B,$4A,$37,$A7,$6E,$FE,$61,$FB),    {
$BD}

($B2,$BB,$87,$75,$B7,$D5,$BF,$59,$C3,$6C,$86,$37,$29,$3A,$46,$02),    {
$BE}

($D6,$E4,$A8,$6E,$03,$B9,$B1,$F0,$BB,$F5,$D9,$58,$2A,$4A,$E8,$EF),    {
$BF}

($48,$43,$A4,$86,$87,$14,$FA,$75,$89,$E8,$EF,$87,$75,$6B,$CA,$CF),    {
$C0}

($19,$4A,$B3,$96,$EB,$68,$EB,$D6,$8D,$47,$62,$85,$B4,$76,$AB,$DA),    {
$C1}

($64,$65,$DA,$D1,$D3,$17,$52,$BE,$3F,$32,$83,$E8,$F7,$0F,$EE,$F7),    {
$C2}

($D7,$82,$76,$F5,$6F,$8E,$C8,$D4,$F8,$CC,$A3,$75,$E4,$53,$43,$66),    {
$C3}
```

```
($FF,$D0,$CE,$5D,$00,$F5,$97,$AA,$4F,$0D,$2B,$B6,$0D,$17,$B6,$BC),    {
$C4}

($EC,$1F,$53,$AA,$1E,$53,$8B,$2C,$3B,$58,$16,$74,$C6,$58,$58,$57),    {
$C5}

($F6,$64,$90,$8B,$48,$B0,$7E,$34,$C3,$47,$2A,$62,$43,$F3,$7C,$BF),    {
$C6}

($56,$DC,$13,$FD,$4D,$31,$F9,$B1,$78,$83,$1A,$27,$4E,$6D,$22,$E2),    {
$C7}

($99,$E3,$A8,$5D,$72,$20,$F8,$6C,$AC,$89,$B6,$7C,$98,$39,$03,$9F),    {
$C8}

($52,$B0,$16,$DB,$15,$C2,$85,$29,$A2,$AA,$2F,$19,$1C,$36,$4D,$FA),    {
$C9}

($CE,$B1,$E7,$DB,$7D,$F6,$A7,$00,$5F,$B5,$B8,$0B,$A6,$E3,$70,$FE),    {
$CA}

($AD,$F8,$DB,$95,$86,$D0,$06,$5D,$23,$6C,$B8,$BF,$50,$BF,$2E,$5F),    {
$CB}

($A2,$E9,$70,$F1,$70,$96,$1C,$E8,$79,$19,$0D,$64,$98,$2C,$94,$EC),    {
$CC}

($E6,$CF,$2A,$A8,$2F,$A3,$71,$B6,$86,$17,$1E,$CD,$6A,$73,$4E,$5D),    {
$CD}

($BD,$A9,$5B,$04,$F2,$96,$87,$64,$A2,$82,$00,$E4,$68,$C7,$7A,$17),    {
$CE}

($55,$12,$D8,$74,$99,$F5,$48,$88,$8C,$6B,$31,$10,$48,$B1,$07,$46),    {
$CF}

($8F,$2C,$5A,$55,$65,$C7,$14,$EB,$71,$19,$4C,$1D,$0B,$91,$4D,$97),    {
$D0}

($AD,$1E,$54,$95,$49,$4F,$17,$68,$00,$3E,$1D,$19,$EE,$D9,$D7,$F5),    {
$D1}

($6B,$AF,$10,$6D,$71,$ED,$92,$D6,$51,$AC,$FB,$2F,$87,$1D,$F4,$A7),    {
$D2}

($7A,$3F,$40,$98,$9A,$17,$8E,$08,$36,$E1,$B2,$FF,$E7,$E2,$F0,$FC),    {
$D3}

($99,$4A,$2D,$20,$77,$EB,$07,$AC,$DE,$B7,$22,$6D,$97,$2B,$2E,$47),    {
$D4}

($D1,$A3,$D5,$01,$C8,$B9,$EC,$19,$9A,$2E,$4C,$93,$F8,$BA,$43,$8D),    {
$D5}

($73,$5E,$75,$A2,$CB,$5E,$F6,$10,$F0,$AE,$B7,$49,$59,$75,$37,$7D),    {
$D6}

($7C,$0B,$80,$29,$C8,$BC,$EB,$E3,$38,$B9,$2C,$EB,$50,$4F,$91,$2A),    {
$D7}

($0C,$78,$AE,$F8,$3F,$66,$AB,$C1,$FA,$1E,$84,$77,$F2,$96,$D3,$94),    {
$D8}

($AE,$6E,$D4,$1F,$37,$D5,$A0,$CD,$AE,$CB,$6B,$E8,$11,$7E,$31,$25),    {
$D9}

($67,$C3,$15,$F6,$14,$5B,$A7,$4C,$EE,$B6,$E5,$DD,$0C,$D1,$B9,$62),    {
$DA}
```

```
($98,$FD,$00,$D7,$88,$AF,$E2,$A5,$FA,$5E,$4F,$8E,$16,$66,$63,$8B),    {
$DB}

($B9,$6A,$24,$BE,$7C,$40,$DC,$85,$F9,$76,$C0,$F8,$66,$B2,$CD,$3B),    {
$DC}

($31,$53,$67,$FD,$7B,$05,$A9,$B7,$97,$A1,$B0,$8F,$4D,$4C,$DF,$BA),    {
$DD}

($AF,$B9,$90,$CB,$F2,$D0,$E0,$A9,$AE,$75,$BA,$2F,$2F,$3B,$B0,$13),    {
$DE}

($A2,$67,$85,$92,$2B,$35,$16,$FE,$62,$7B,$AB,$97,$26,$C6,$6E,$43),    {
$DF}

($EC,$2D,$11,$02,$87,$66,$E0,$6A,$C3,$36,$48,$E2,$F0,$A6,$73,$20),    {
$E0}

($2E,$EC,$E4,$37,$6C,$EE,$14,$33,$D0,$E9,$F2,$00,$DE,$B7,$54,$08),    {
$E1}

($D8,$1F,$D9,$B2,$6F,$D0,$E8,$9A,$61,$B6,$52,$29,$AA,$C2,$86,$E5),    {
$E2}

($69,$B7,$A7,$30,$8E,$E1,$B0,$65,$AA,$30,$8E,$63,$C4,$4A,$E0,$F3),    {
$E3}

($C1,$5B,$CC,$55,$77,$F9,$FA,$DE,$4B,$4A,$32,$56,$19,$0A,$59,$B0),    {
$E4}

($73,$20,$33,$61,$CC,$9B,$ED,$58,$2F,$2D,$0E,$AE,$A0,$99,$CB,$38),    {
$E5}

($31,$55,$1A,$3A,$E5,$9F,$05,$96,$D9,$67,$B3,$F7,$57,$66,$28,$94),    {
$E6}

($9D,$5A,$27,$3E,$0F,$B6,$AE,$BC,$A8,$25,$00,$9E,$E2,$36,$3E,$2C),    {
$E7}

($78,$5D,$51,$2B,$E4,$31,$6D,$57,$8E,$66,$50,$61,$3B,$45,$E9,$34),    {
$E8}

($34,$06,$87,$76,$94,$69,$1D,$DD,$1D,$FB,$0A,$CA,$54,$68,$14,$07),    {
$E9}

($86,$27,$7F,$69,$03,$50,$A0,$8D,$08,$05,$5C,$BD,$DA,$22,$5E,$0A),    {
$EA}

($AB,$95,$F1,$FA,$7D,$A6,$A9,$02,$74,$40,$9B,$89,$56,$2F,$3F,$FD),    {
$EB}

($16,$7B,$86,$F2,$1D,$F3,$76,$C9,$6F,$10,$A0,$61,$5B,$14,$20,$0B),    {
$EC}

($26,$FC,$1F,$29,$BD,$45,$99,$F5,$F2,$92,$00,$B6,$CA,$08,$35,$31),    {
$ED}

($FC,$12,$62,$74,$64,$24,$40,$22,$78,$E8,$8F,$6C,$1F,$02,$F5,$81),    {
$EE}

($25,$75,$07,$9E,$53,$E0,$60,$5B,$24,$B1,$BD,$8D,$F2,$E2,$F7,$57),    {
$EF}

($8C,$49,$3A,$43,$D8,$C1,$EF,$79,$88,$60,$BB,$02,$B6,$2E,$8E,$79),    {
$F0}

($ED,$B9,$07,$36,$12,$19,$FB,$8D,$50,$27,$9E,$AB,$AB,$0B,$83,$B1),    {
$F1}
```

```
($7A,$94,$05,$D4,$59,$C2,$A9,$28,$B1,$29,$52,$E2,$76,$F9,$A8,$F5),    {
$F2}

($63,$34,$C2,$AE,$05,$C2,$42,$1C,$68,$7F,$51,$67,$72,$B8,$17,$DA),    {
$F3}

($97,$A6,$DD,$4C,$45,$B2,$3D,$B9,$C5,$D6,$03,$CE,$16,$1B,$8C,$AB),    {
$F4}

($AD,$E7,$A0,$DC,$F4,$DD,$C0,$67,$3E,$D4,$8B,$70,$A4,$A3,$40,$D6),    {
$F5}

($44,$F2,$C5,$0B,$83,$8F,$92,$97,$0D,$2C,$11,$E2,$FA,$98,$BC,$7D),    {
$F6}

($19,$32,$A6,$84,$9C,$6F,$57,$5C,$A3,$60,$26,$6C,$C3,$F9,$A4,$66),    {
$F7}

($31,$74,$16,$35,$B4,$1D,$53,$50,$98,$24,$1F,$EA,$03,$C1,$E4,$7F),    {
$F8}

($89,$DE,$FC,$50,$A7,$0E,$A5,$61,$7C,$2D,$4B,$BF,$E0,$CA,$5C,$DE),    {
$F9}

($89,$01,$CF,$EA,$DA,$A3,$DA,$45,$D4,$AF,$4C,$40,$2E,$FB,$4D,$4E),    {
$FA}

($EA,$8F,$D1,$AA,$FD,$5A,$33,$09,$77,$A2,$E1,$2E,$6F,$F9,$86,$D5),    {
$FB}

($CF,$0E,$EC,$E3,$A2,$3B,$68,$0F,$62,$66,$A2,$1A,$AB,$BA,$4D,$32),    {
$FC}

($DA,$56,$4F,$38,$41,$3A,$24,$3E,$30,$E8,$C8,$C0,$7F,$CC,$C5,$D8),    {
$FD}

($40,$3A,$E0,$91,$D3,$BE,$6A,$CF,$11,$81,$14,$85,$27,$F1,$E0,$AE),    {
$FE}

($00,$59,$4F,$D4,$F4,$2B,$A4,$3F,$C1,$CA,$04,$27,$A0,$57,$62,$95)    {
$FF}

       );
```

En el listado 1-1 se encripto cada uno de los valores hexadecimales desde ‘’$00 hasta FF” lo cual nos permite comparar carácter a carácter de las entradas y buscar dentro de esta tabla cual es el hash que corresponde para cada uno de estos.

Verificación de errores

Listado 9-2 bucle principal

```
for counter  :=$00 To $FF
   do
   texto_hash :=MD5 (counter);
end.
```

9.1.1 Verificación de errores

Listado 9-3 bucle principal

```
for counter  :=$00 To $FF
   do
   texto_hash :=MD5 (counter);
end.
```

En el listado 1-2 se puede ver le bucle principal donde se tiene la variable “counter” el cual toma valores desde $00 a $FF, y lo cual para cada uno de estos valores llama a la función “MD5”, y esta función es la que contiene el programa encargado de encriptar las palabras que mostramos anteriormente en la Listado 9-4 sección de desarrollo del algoritmo.

Listado 9-3

```
Var Clave, conver :String ;

    texto_hash: String ;
    data_chr  : Char ;
    data_hexa : byte ;
    tabla :  array [0..255] of String  ;
    counter   : word ;
```

Después definir la tabla se definen variables globales donde se muestran “clave, conver” como variables que almacenan string los cuales servirán para almacenar semillas a encriptar. También se definen la variable “texto_hash” variable que almacena lo que hay en la función “MD5” en el bucle principal. Las variables “data_chr” se ocupará para almacenar variables del tipo char dentro del programa, la variable “data_hexa” que es del tipo byte se almacenaran byte por byte los HASH en hexadecimal. La variable “tabla” es un array que almacena los 256 valores hexadecimal. La variable “counter” de tipo Word en el bucle principal es utilizado como un contador para el argumento de la función “MD5”.

Listado 9-5 Función “MD5”

```
Function md5 (x:word ):string;
var a:array[0..15] of byte;
    s : string ;
    i:integer;
    dato_binar : word ;
    dato_ascci : Char Absolute dato_binar;
 LenHi, LenLo: longint ;
 Index: longint;
 HashBuffer: array[0..63] of byte;
 CurrentHash: array[0..3] of longint ;
```

A esta función se les suma a las variables ya definidas anteriormente las variables “dato_binar” que se ocupará para guardar los datos binarios del programa, y “dato_ascci” que almacenará los datos el char de binario.

Listado 9-6 bucle función “MD5”

```
Begin
 Init;
 dato_binar := x ;
 s := dato_ascci ;

 Update(s[1],Length(s));


 Final(a);

for i:=0 to 15 do
  BEGIN
   if ( a[i]  <> enig [x][i]) Then  writeln (‘ error [’,x,’ ‘,i,’]’);
  End;
 Burn;
End;
```

En este bucle se ejecuta el procedimiento “init” y se almacenan los valores de x dentro de la variable “dato_binar”, y en la variable “s” se almacenan los valores de “dato_ascci” el cual llamará al procedimiento “update” apuntado al primer espacio de la variable “s” y tomando el largo del string.

Al ejecutar el procedimiento final se obtendrá un resumen de HASH de 16 byte los cuales serán almacenado en la variable “a”, se utiliza la variable “i” para hacer un contador que recorra cada uno de los 16 byte de la salida que es el HASH de cada carácter hexadecimal encriptado en el argumento de la función “MD5”, y en bucle de “for” se busca cada uno de estos bytes en el arreglo si está correcta la tabla en cualquier caso de que el arreglo presente algún error de coincidencia con los HASH de salida, se mostrará por pantalla la fila y la columna donde se presenta el error, con esto se asegura el éxito en los siguientes pasos del proyecto.

9.1.2 Prueba de errores

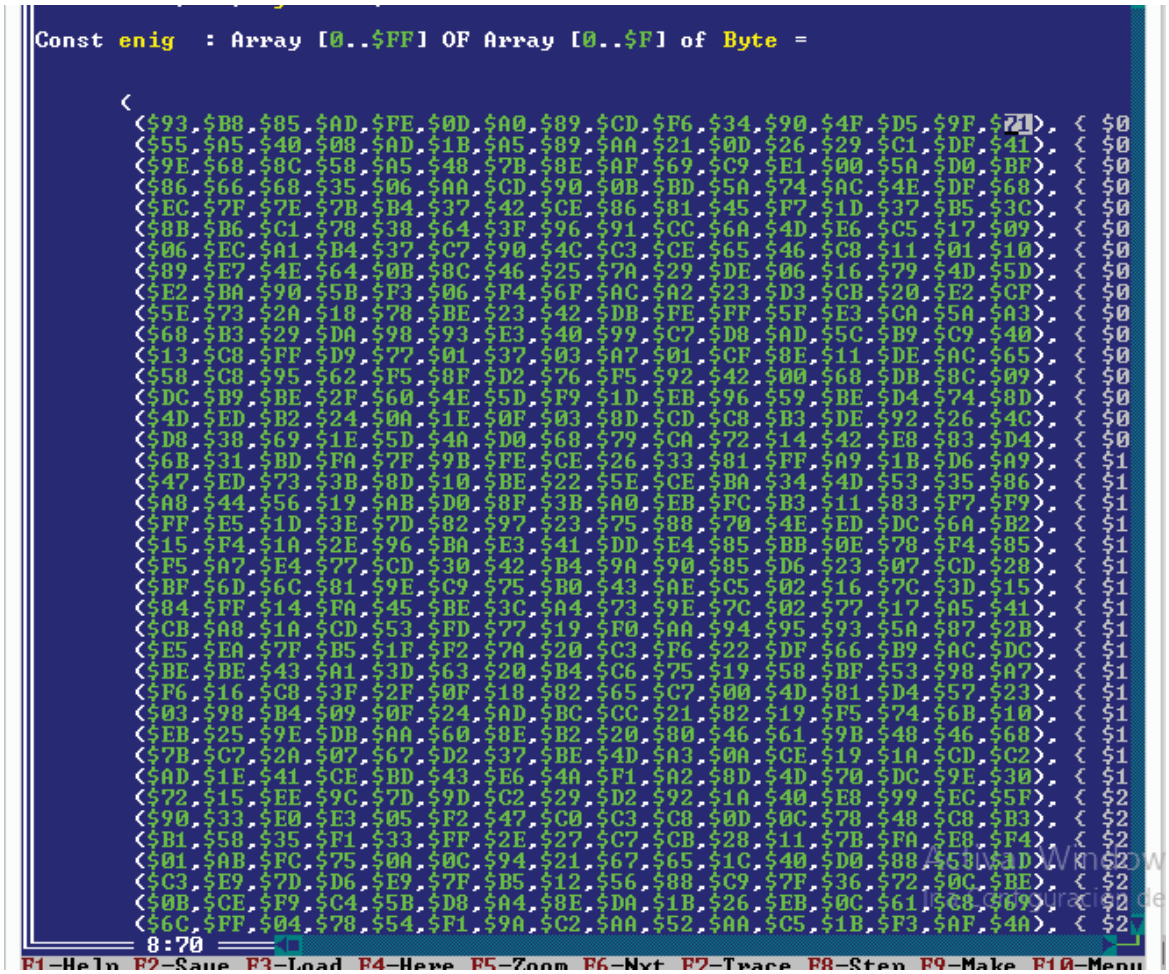


Ilustración 28 Arreglo original

En la ilustración 29 se muestra el arreglo “enig” original y se seleccionó el 71 el cual será modificado por el número 72, con el objetivo que al arrancar el programa se pueda visualizar un error y que indique ese espacio del arreglo que presenta dificultades.

Ilustración 29 Prueba de tabla original

Al compilar el programa no se registra error alguno lo que verifica que los valores de la tabla hexadecimal están correctos, lo cual se contradice con el contraejemplo que se muestra en la figura 9-3 mostrada anteriormente.

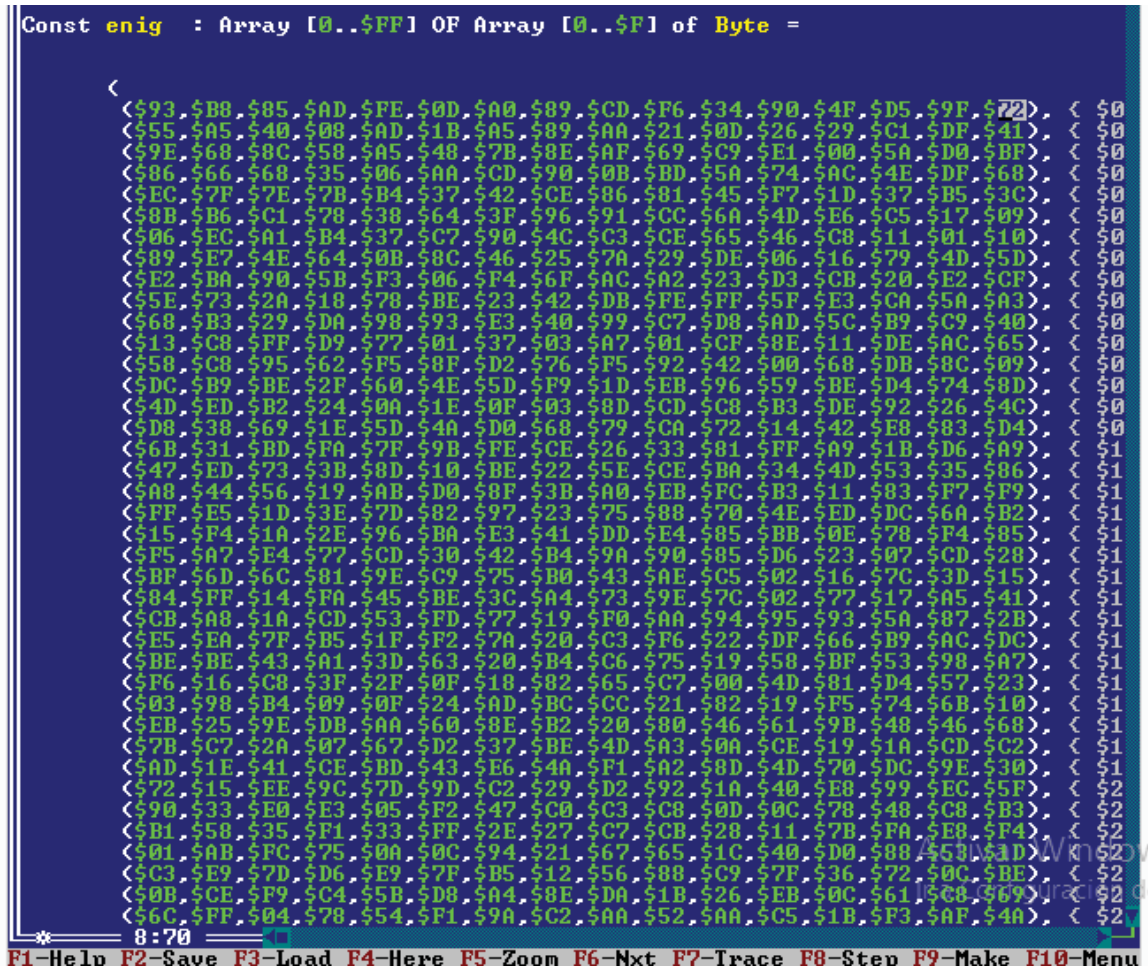


Ilustración 30 Arreglo "enig" modificado

En la ilustración 31 se muestra seleccionado el número del arreglo que se modificó, con esto se puede hacer la prueba de error para lo cual se debe arrancar el programa.

Ilustración 31 Resultado prueba

En la figura se imprimen el lugar exacto donde se encuentra el error dentro de la matriz que contiene cada uno de los hashes para cada uno de los valores de la tabla en hexadecimal.

9.2 Archivos

Virtual pascal tiene la opción de leer archivos y operarlo según las necesidades del usuario.

9.2.1 Lectura de archivos

Listado 9-6 Lectura archivo

```
program lectura_archivo ;
var
 F1: text;
 caracter: char;

 begin
 Assign (F1,'C:\Users\Matias\Desktop\entrada_salida\entrada.txt');
 Reset (F1);
while not Eof (F1) do
begin
        while not Eoln (F1) do
        begin
                read(F1, caracter);
                write(caracter);
        end;
end;
close (F1);
Readln;
end.
```

En el listado 1-6 se muestra la lectura de un archivo, se definieron las variables “F1” de texto, y la variable “carácter” de tipo “char” , en el bucle principal se muestra el comando “Assign” donde se le asigna a la variable “F1” lo que se encuentra en la dirección del archivo a la que se apunta

dentro del comando " 'C:\Users\Matias\Desktop\entrada_salida\entrada.txt')", el procedimiento "Reset" abre un nuevo archivo existente para una operación de lectura en este caso para "F1" .

Dentro del bucle del programa con la condición "while not Eoln (F1) do" lee hasta el final de cada línea del archivo y en el bucle del while se escribe en la variable "carácter" lo almacenado en F1, y se imprime por pantalla, para luego terminar la condición terminar la lectura.

Ilustración 32 Contenido archivo de texto

La ilustración muestra el contenido a leer por el programa.

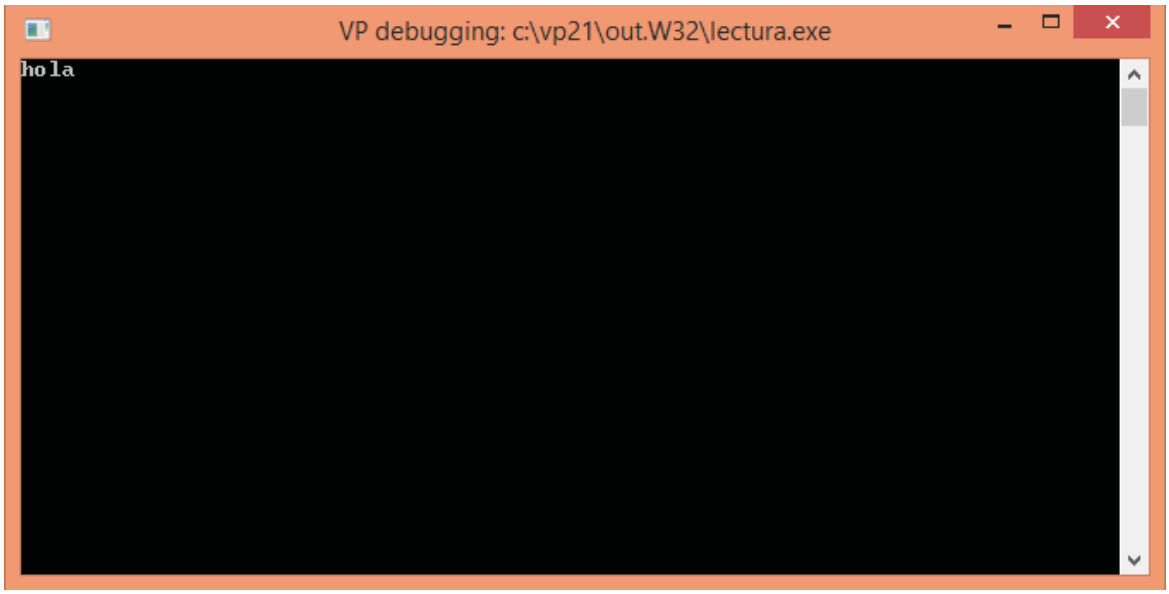


Ilustración 33 Impresión lectura de archivo

9.2.2 Transferencia de archivos

Se puede leer un archivo, pero a su vez se pueden transferir a otro archivo lo cual es muy necesario para el desarrollo de este proyecto.

Listado 9-7 Transferencia de archivos

```
program Transferencia_archivos ;
var
 F1, F2: text;
 cadena: string;

 begin
 Assign (F1,'C:\Users\Matias\Desktop\entrada_salida\entrada.txt');
 Assign (F2,'C:\Users\Matias\Desktop\entrada_salida\salida.txt');
 Reset (F1);
 Rewrite(F2);
   while not Eof (F1) do
   begin
      while not Eoln (F1) do
      begin
         read(F1, cadena);
         write(F2, cadena);
      end;

      Readln(F1);
      writeln(F2);
  end;
 close (F1);
 close (F2);
end.
```

Se puede leer un archivo, pero a su vez se pueden transferir a otro archivo lo cual es muy necesario para el desarrollo de este proyecto.

Se declaran las variables “F1” y “F2” que contienen archivos de texto y “cadena “que contiene un string, dentro del bucle se asigna a la variable F1 el contenido del archivo de texto “’C:\Users\Matias\Desktop\entrada_salida\entrada.txt’” para “F2” se le asigna el contenido de la dirección “C:\Users\Matias\Desktop\entrada_salida\salida.txt”, con el procedimiento “Rewrite(F2) “ se crea y abre un archivo si el archivo ya existe borra su contenido, en caso contrario el archivo “F2” queda abierto para una operación de escritura, después se encuentra la operación “while not Eof (F1) do” donde “Eof” es una función de tipo lógico que indica si el fin del archivo se ha almacenado y devuelve “true” si se ha almacenado y false en el caso contrario. Entonces mientras no se llegue al final del archivo “F1” se seguirá ejecutando lo que está dentro del bucle de la sentencia “while do”, hay un segundo “while do” donde “read (F1, cadena)” escribe en la variable cadena lo almacenado en “F1” para luego escribirlo en “write (F2, cadena)” línea por línea. Asi finaliza el segundo “while do” y mostrar por pantalla lo almacenado en “F2” que corresponde al archivo de texto “F1” finalizando la primera sentencia y cerrando “F1” y “F2” y así terminado el programa.

9.3 Transmisión byte a byte

Al trasmitir un archivo byte a byte se busca tener una archivo en la entrada que contenga una cantidad indefinida de carácter, los cuales para cada carácter corresponde un byte, los cuales serán enviado a la matriz que contiene los HASH de todos los valores hexadecimal y al encontrar los hash de cada uno de los carácter serán enviado a un archivo de salido donde se almacenaran en orden uno por uno estos HASH, lo que implica que si un archivo de entrada tiene un tamaño de 4 byte a la salida se obtiene un archivo de 64 byte.

9.3.1 Programa transmisión byte a byte

Listado 9-8 Transferencia byte por byte

```
for counter  :=$00 To $FF do
begin
        texto_hash :=MD5 (counter);
end;

Assign (F1,'C:\Users\Matias\Desktop\entrada_salida\entrada.txt');
Assign (F2,'C:\Users\Matias\Desktop\entrada_salida\salida.txt');
Reset (F1);
Rewrite(F2);
while not Eof (F1) do
begin
        while not Eoln (F1) do
        begin
                read(F1, caracter);
                for counter := $00 To $F do
                begin
                        write(F2,chr(enig[ord(caracter)][counter]));
                end;
        end;
end;
close (F1);
close (F2);

readln;

end.
```

Para lograr la transmisión de datos byte por byte se requiere ocupar el programa de transmisión de archivos como eje principal, donde se tiene una dirección de archivo que se asigna a la variable “F1” que es de donde se sacaran los carácter que se transmitirán una vez encontrada su encriptación en la matriz “enig”, los cuales serán almacenado línea a línea en “F2”, donde “Reset(F1)” sigue abriendo el archivo para su lectura, y “Rewrite ” borra el contenido del archivo y le deja abierto para el proceso de escritura y se ocupan las líneas antes nombradas “while not Eof (F1) para que se recorran todos los datos del archivo “F1” antes que se deje de ejecutar lo que está dentro de la segunda sentencia do”“while not Eoln (F1) do” que recorre línea a línea lo que está en “F1” y ejecuta lo que está dentro del bucle de la segunda sentencia “while do” así con “read(F1, caracter)” se escribe en carácter lo que está en “F1” se ejecuta “for counter := $00 To $F do” donde counter toma 16 valores en hexadecimal y para cada uno de esos valores se ejecuta “write(F2,chr(enig[ord(caracter)][counter]))” que escribe en el archivo F2 el valor en ASCII del hash correspondiente al número en decimal a la fila de la matriz “enig” así carácter por carácter encuentra su hash correspondiente y lo envía a “F2, luego se cierra la variable “F1” y “F2” y se le da fin al procedimiento.

9.3.2 Prueba transmisión de archivo byte a byte

Se tiene un archivo de entrada que mantendrá las características de los ejemplos anteriores.

Ilustración 34 Entrada de texto

Se tienen las caracteristicas de los datos de entrada a transmitir encriptados con MD5 byte por byte.

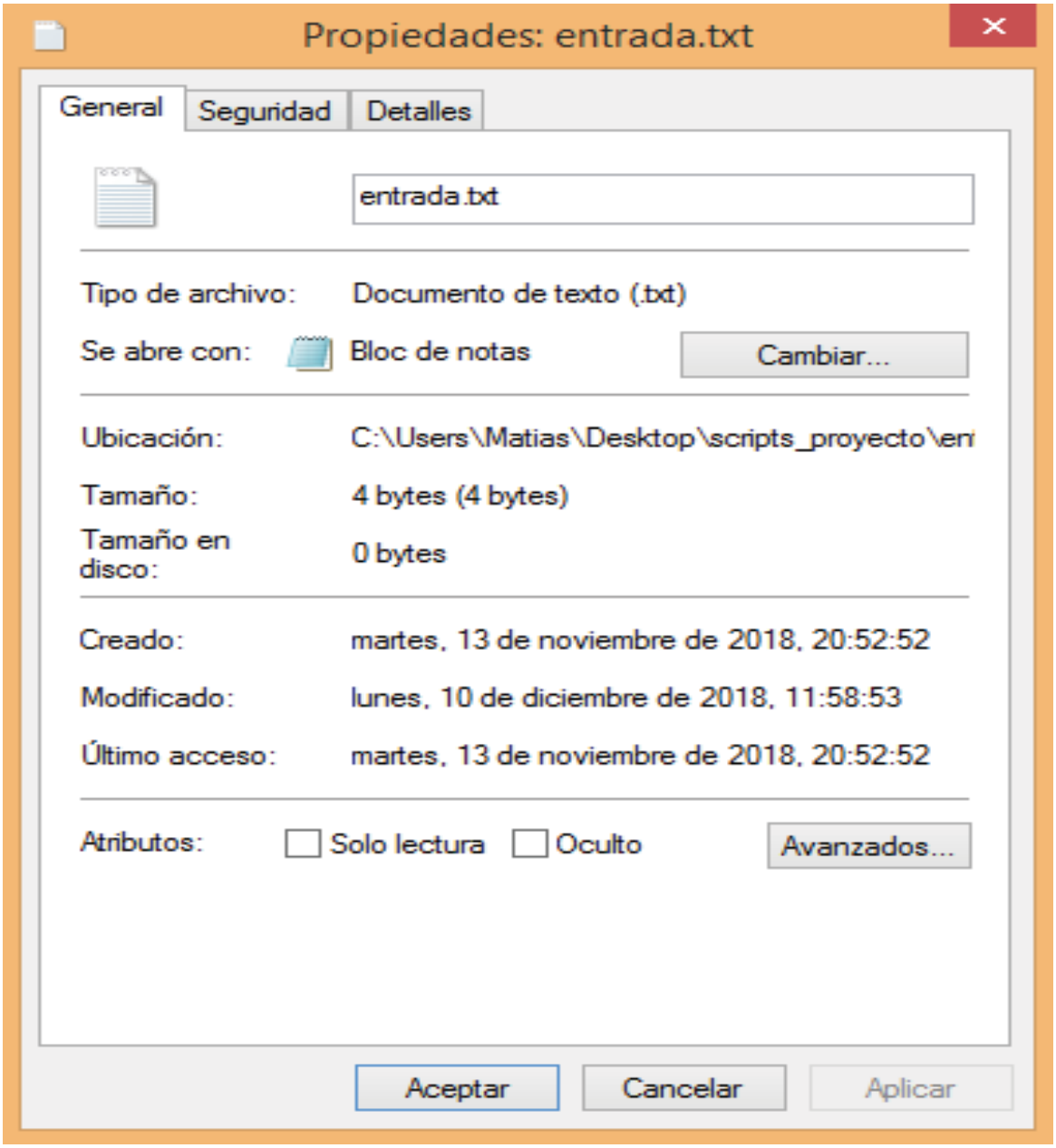


Ilustración 35 Características archivo de entrada

Se aprecia en la ilustración que el tamaño de la entrada es de 4 byte, esto significa que en la salida debemos tener un archivo de tamaño 64 byte donde cada 16 byte corresponderá al carácter de entrada en orden de ingreso.

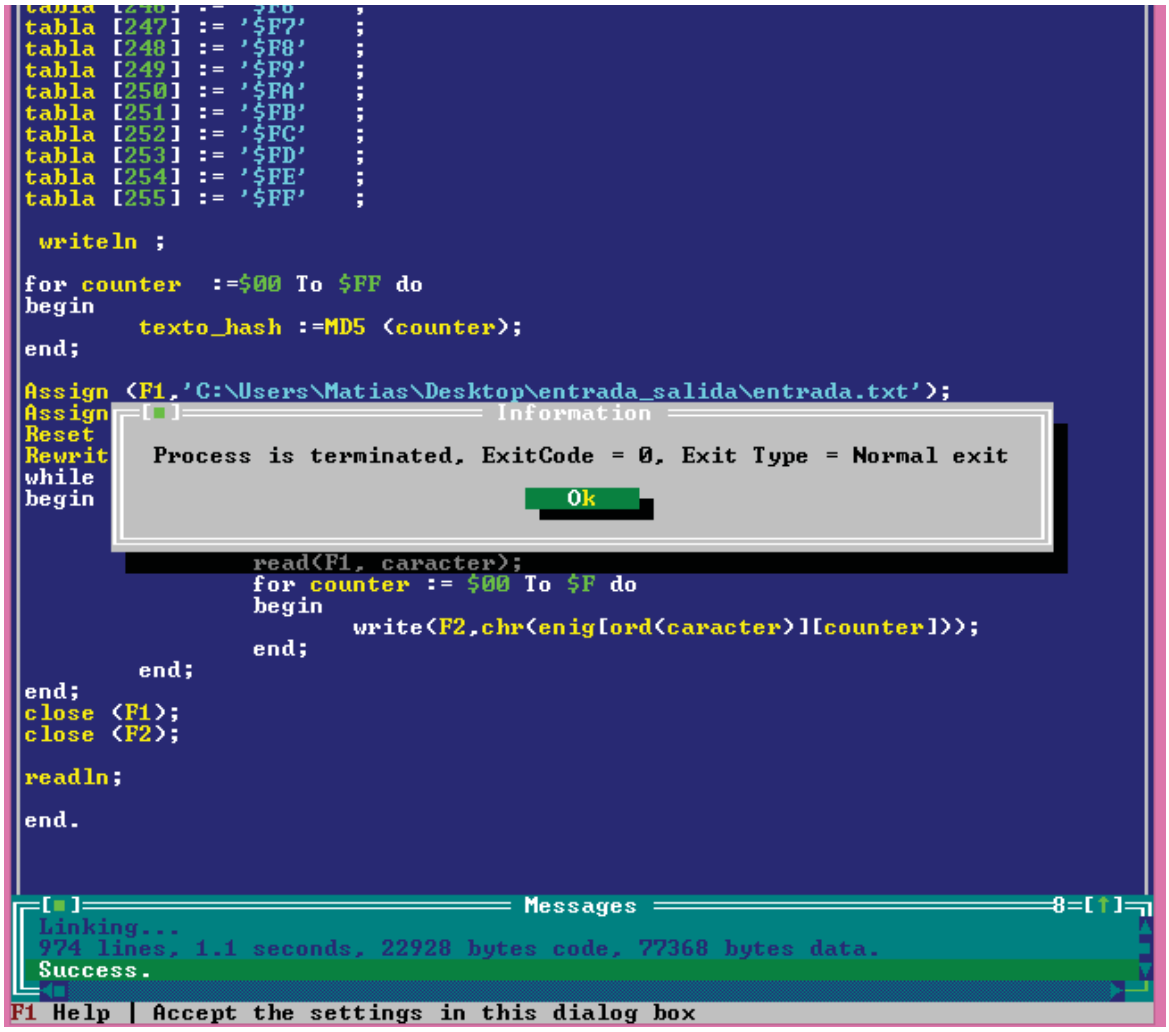


Ilustración 36 Compilación programa

Al compilar y arrancar programa no presenta errores y se asume que los datos que se encuentran en el archivo de salida son los correspondiente a “hola” carácter por carácter, pero transmitido su HASH en ASCII.

Entonces los resultados de la salida son los siguientes.

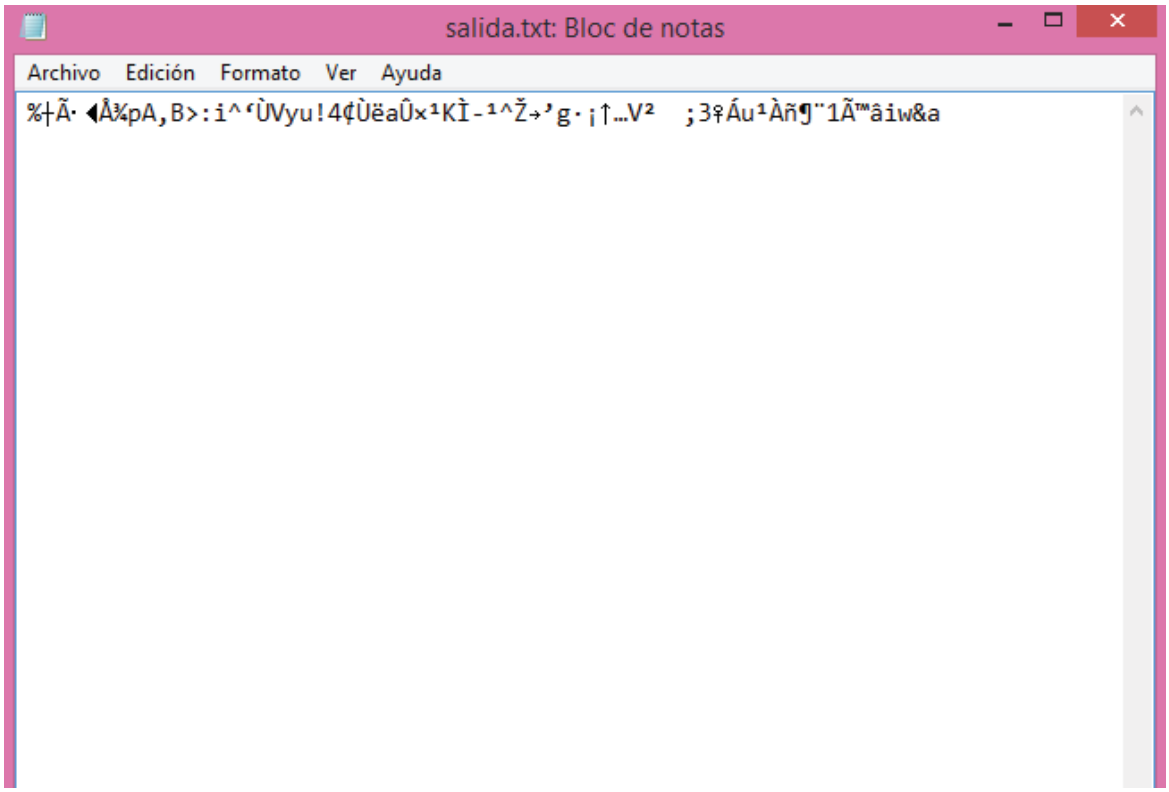


Ilustración 37 Resultados salida

En la ilustración 38 se puede apreciar que en el archivo de texto salida tenemos los hashes correspondientes a “hola”, de los cuales los primeros 16 byte corresponden a la letra “h” y los 16 byte siguientes a la segunda letra y así sucesivamente, logrando así que el tamaño del archivo de salida sea de 64 byte.

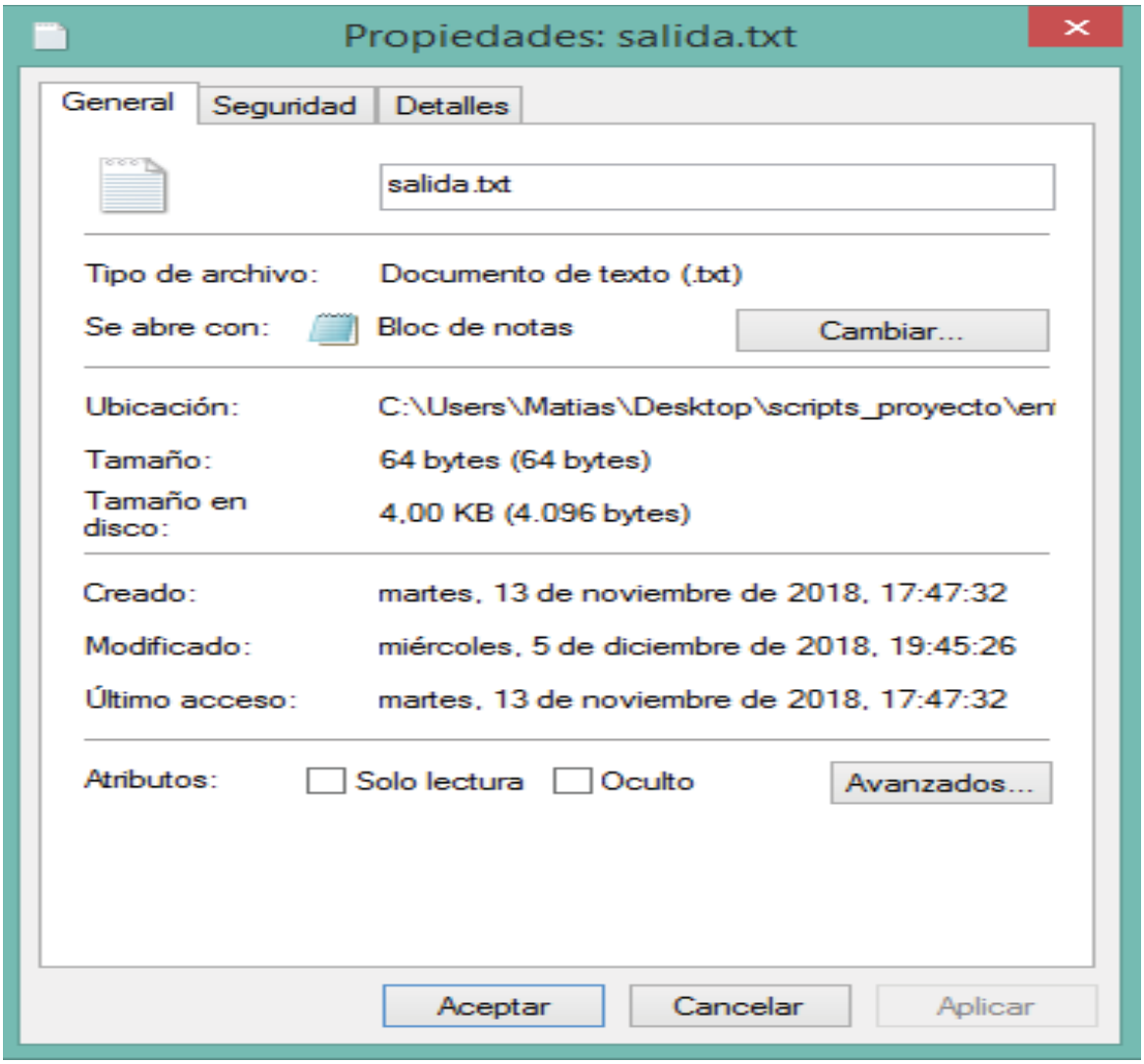


Ilustración 38 Características salida

En las propiedades del archivo se puede apreciar que el tamaño del archivo es de 64 byte lo que cumple para nuestra entrada, por lo cual tiene almacenado 4 hash de 16 byte cada uno que corresponde a cada uno de los caracteres.

10 Tiempo de encriptado

En el listado 10-1 se muestra cómo se puede tomar el tiempo de encriptación de las semillas en la entrada, poniendo un timer “tiempo1: = sec100+second*100;” el cual tomará el tiempo antes de encriptar las palabras. Se ocupará un segundo timer para tomar el tiempo al final de la encriptación, “tiempo2: = sec100+second*100”, después se almacenará en” tiempo_total”. La diferencia de los dos tiempos que será el tiempo total de encriptación. Este tiempo se mostrará por pantalla, pero al ser este muy pequeño se multiplica por 100 para hacer legible el resultado.

Listado 10-1 Timer

```
Begin
Init;
Update(s[1],Length(s));
Final(a);
{result:=”;}
for i:=0 to 15 do
  BEGIN
   write (A[I]);
   WRITE (‘ ‘);
 END;
Burn;
End;

begin
  GetTime( Hour, Minute, Second, Sec100 );
  tiempo1:=sec100+second*100;
  texto_hash := MD5 (‘algoritmo’);
  GetTime( Hour, Minute, Second, Sec100 );
  tiempo2:=sec100+second*100;
  tiempo_total:= tiempo2-tiempo1;
  writeln (‘Tiempo:=  ‘,tiempo_total*10);
 // writeln(tiempo2-tiempo1);
  writeln (texto_hash);
 { writeln ( MD5 (‘HOLA’));}
```

10.1 Pruebas

En esta sección se hicieron pruebas para calcular el tiempo, principalmente se realizó colocando un timer que almacenará el momento exacto donde se empieza a encriptar una palabra y luego con un segundo temporizador se muestre el instante donde termina el encriptado, luego haciendo una comparación y calculando el delta del tiempo se puede saber con exactitud cuál es el tiempo de encriptado.

10.1.1 Prueba 1

```
procedure Final(var Digest);
begin
  HashBuffer[Index]:= $80;
  if Index>= 56 then Compress;
  mem  [ofs (HashBuffer[56])] := LenLo;
  mem  [ofs (HashBuffer[60])] := LenHi;
  Compress;
  Move(CurrentHash,Digest,Sizeof(CurrentHash));// Digest es un array de 16 byt
  Burn;// devuelve todo a cero todo lo que no esta definido dentro de currenth
end;


begin
Init;
Update(s[1],Length(s));
Final(a);
{result:='';}
for i:=0 to 15 do
  BEGIN
   write (A[I]);
   WRITE (' ');
 END;
Burn;
End;

begin
  GetTime( Hour, Minute, Second, Sec100 );
  tiempo1:=sec100+second*100;
  texto_hash := MD5 ('algoritmo');
  GetTime( Hour, Minute, Second, Sec100 );
  tiempo2:=sec100+second*100;
  tiempo_total:= tiempo2-tiempo1;
  writeln ('Tiempo:=  ',tiempo_total*10);
 // writeln(tiempo2-tiempo1);
  writeln (texto_hash);
 { writeln ( MD5 ('HOLA'));}

end.
      410:70
F1-Help F2-Save F3-Load F4-Here F5-Zoom F6-Nxt F7-Trace F8-Step F9-Make F10-Menu
```

Ilustración 39 Entrada "algoritmo"

Se tiene en la ilustración 40 una entrada de 9 caracteres a continuación se calcular el tiempo de esta palabra con MD5.

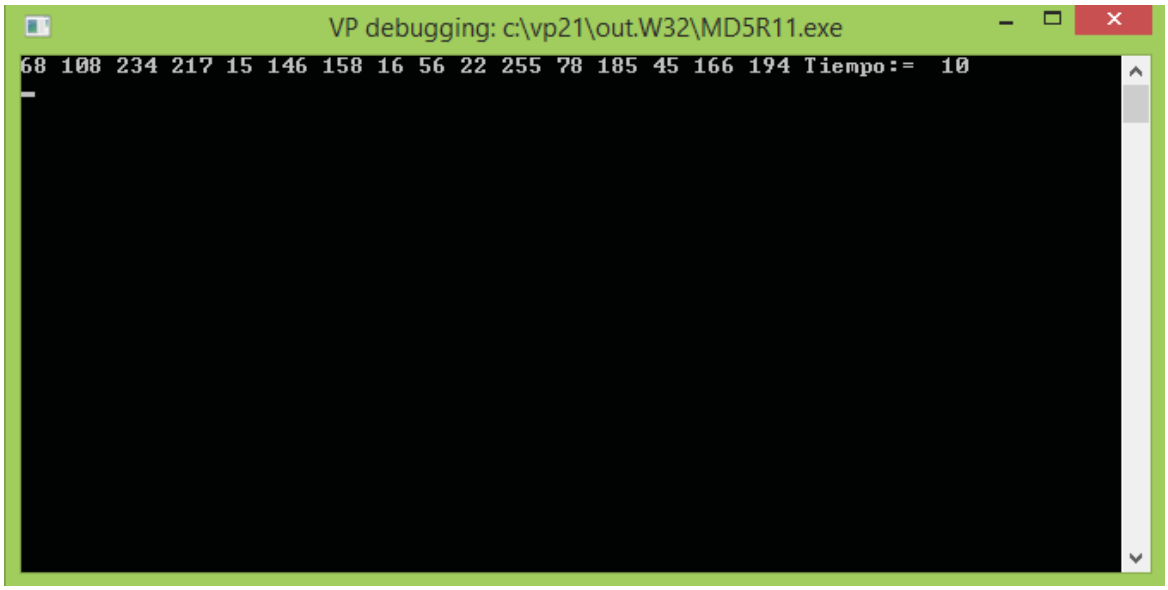


Ilustración 40 Tiempo semilla "algoritmo"

En la ilustración 41 nos muestra el tiempo de encriptación de la semilla algoritmo, el cual es de 10 centésimas de segundo.

10.1.2 Prueba 2

```
procedure Final(var Digest);
begin
  HashBuffer[Index]:= $80;
  if Index>= 56 then Compress;
  mem  [ofs (HashBuffer[56])] := LenLo;
  mem  [ofs (HashBuffer[60])] := LenHi;
  Compress;
  Move(CurrentHash,Digest,Sizeof(CurrentHash));// Digest es un array de 16 byt
  Burn;// devuelve todo a cero todo lo que no esta definido dentro de currenth
end;


begin
Init;
Update(s[1],Length(s));
Final(a);
{result:='';}
for i:=0 to 15 do
  BEGIN
   write (A[I]);
   WRITE (' ');
 END;
Burn;
End;

begin
  GetTime( Hour, Minute, Second, Sec100 );
  tiempo1:=sec100+second*100;
  texto_hash := MD5 ('paralelepipedo');
  GetTime( Hour, Minute, Second, Sec100 );
  tiempo2:=sec100+second*1000;
  tiempo_total:= tiempo2-tiempo1;
  writeln ('Tiempo:=  ',tiempo_total*10);
 // writeln(tiempo2-tiempo1);
  writeln (texto_hash);
 { writeln ( MD5 ('HOLA'));}

end.
      397:78
F1-Help F2-Save F3-Load F4-Here F5-Zoom F6-Nxt F7-Trace F8-Step F9-Make F10-Menu
```

Ilustración 41 Entrada "paralelepipedo"

Para la entrada “paralelepípedo” el tiempo de encriptado es:

```
VP debugging: c:\vp21\out.W32\MD5R11.exe
219 167 37 20 162 10 7 191 33 217 229 157 25 166 87 112 Tiempo:=  279000
```

Ilustración 42 Tiempo de "paralelepipedo"

El tiempo de encriptado que nos entrega la pantalla en la ilustración 43 es de 279 milésimas de segundos, en comparación a la prueba uno esta palabra fue más rápida de encriptar.

10.1.3 Prueba 3

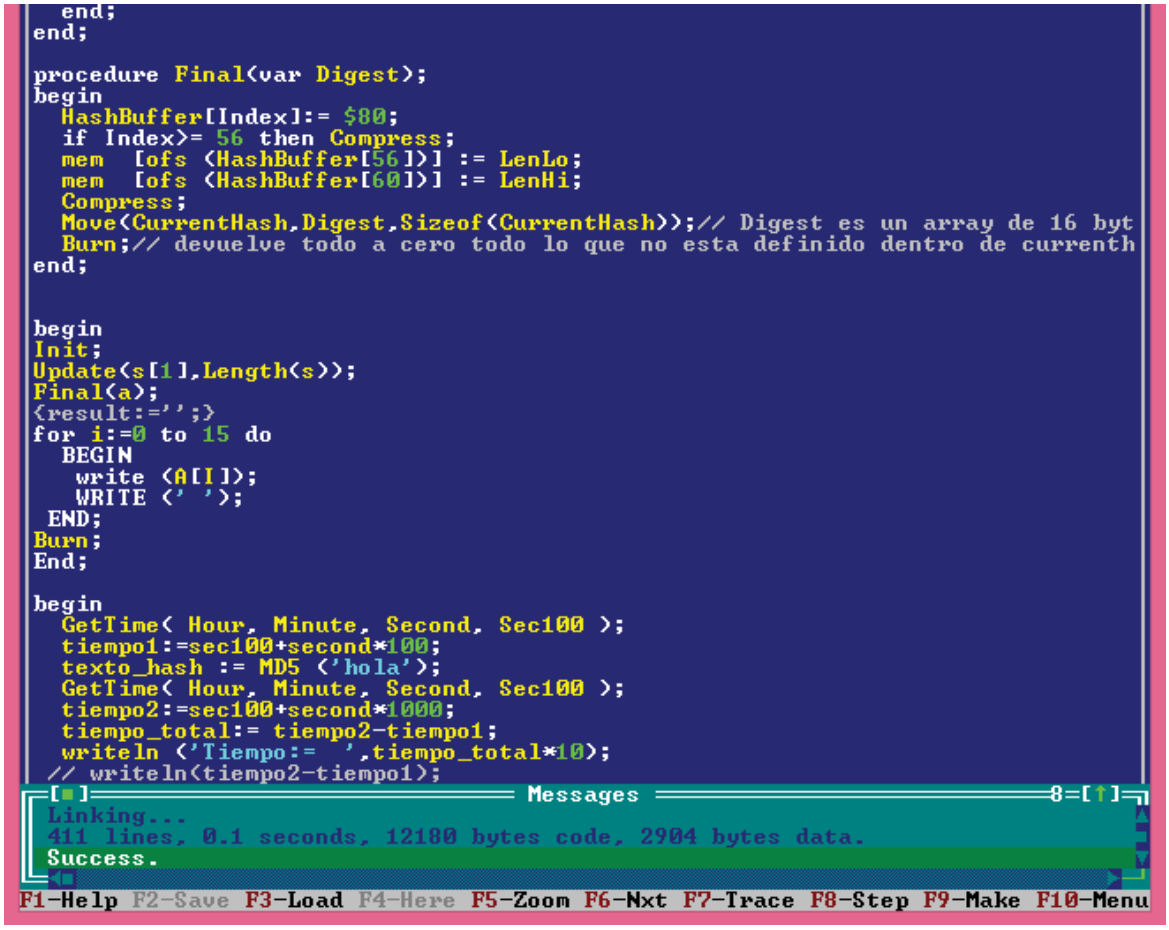


Ilustración 43 Entrada "hola"

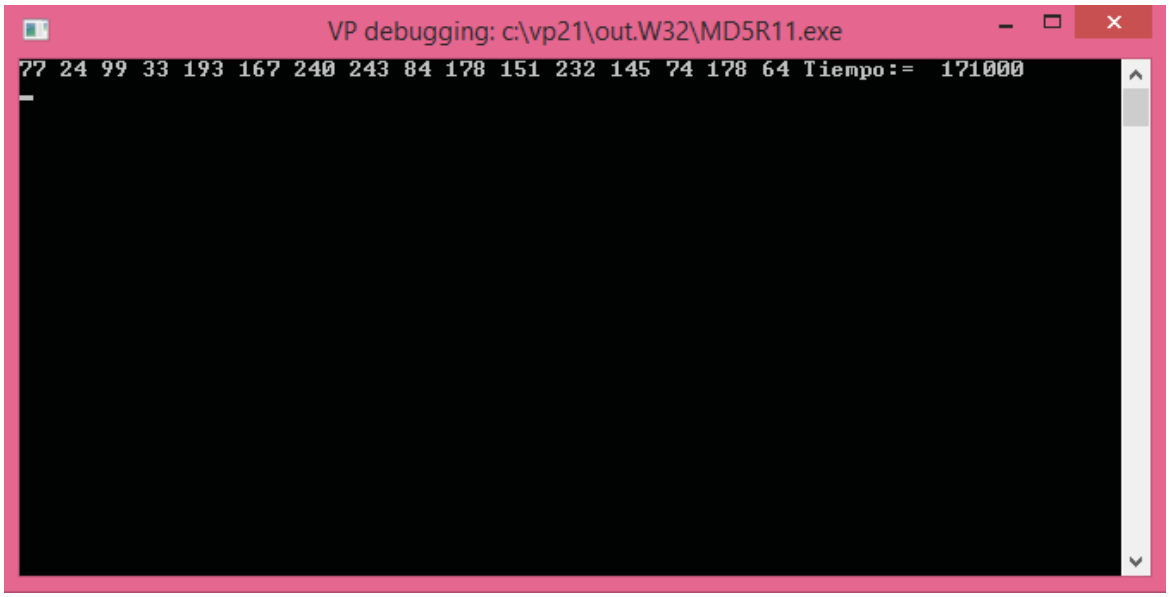


Ilustración 44 Tiempo de entrada "hola"

En la pantalla se muestra que la encriptación tuvo un tiempo de 171 milésimas de segundo lo cual es más rápido que los ejemplos anteriores y se puede verificar que su grado de dificultad es menor.

Discusión y conclusiones

La implantación de un verificador algoritmo MD5 no es una tarea sencilla, por lo que se debe estudiar a fondo cada lenguaje en los que ya está implementado teniendo en cuenta cual será el más conveniente para desarrollar el proyecto en cuestión. Esta tarea se complica aún más teniendo en cuenta que dichos programas son totalmente nuevos y diferentes entre sí, pero al trabajar y adquirir el conocimiento necesario se puede trabajar de una forma más eficiente sobre el algoritmo.

Dentro de las propuestas de trabajo, los programas postulados son los más adecuados por el parecido en su estructura y metodología de programación, OS, que a medida que avanzó el proyecto logro ser muy parecida a la del programa final en su estructura.

También se puede concluir en base a las pruebas realizadas dentro de este proyecto que MD5 es un algoritmo que ha demostrado ser bastante óptimo en su funcionamiento lo cual genera una gran motivación para seguir desarrollándolo, pero tomando en cuantos ciertos aspectos de lo que significa entrar en este campo. Desarrollar este tipo de algoritmos conlleva una gran responsabilidad dado que la seguridad informática hoy en día es muy importante, y que existe un número muy grande de amenazas en la red que pueden ser perjudiciales para las instituciones y sus usuarios. Así el objetivo de este proyecto fue asegurar que el intercambio de información se realice con la mayor seguridad posible.

Por otro lado, siempre en bueno indagar en otros tipos de algoritmos de encriptación, aunque la orientación principal fue la del trabajo sobre MD5, se hizo necesario una vez avanzado el proyecto profundizar en SHA, esto le da un valor agregado al tener actualizaciones que hasta el día de hoy no han sido vulneradas, dando a conocer sus aplicaciones y soporte de cada una de estas. Con esto se cumple el objetivo de dar al lector la mayor cantidad de información necesaria para que pueda evaluar si el algoritmo desarrollado cumple con sus expectativas de trabajo.

Por otro lado, se cumple con el objetivo planteado en la primera Mesa redonda, que es definir los lenguajes de programación que se van a utilizar, y en base a lo estudiado nos garantizara el éxito a la hora del desarrollo del algoritmo MD5.

También concluir que se cumplió con el objetivo principal de esta sección, y se obtuvo éxito en las pruebas de encriptación de entradas aleatorias de largo indefinido, mostrando resultados comparativos en la salida una salida alfanumérica de largo definido para cualquier tipo de

entrada, no obstante, se plantean muchas más pruebas por posibles vulnerabilidades que se podrían llegar a tener, así se puede seguir proporcionando información del algoritmo MD5.

En el ámbito del lenguaje se cumplió con la explicación de la estructura lógica del programa dando a conocer cada uno de sus elementos, y que función cumple dentro del programa como los son las funciones y procedimientos, los cuales están en orden de ejecución dentro del programa para así con las opciones del compilador virtual pascal se pueda verificar los resultados y valores paso a paso del programa y en caso de que exista encontrar errores.

A los objetivos generales se puede señalar que se cumplen al seguir desarrollando a través de pruebas impulsadas principalmente por la necesidad de visualizar en tiempo real cada uno de los posibles errores que puede presentar el programa a la hora de encriptar, esto nos permitió controlar cada uno de los parámetros del programa así garantizando una correcta encriptación y con errores nulos, corroborando una encriptación exitosa para todos los casos.

Por otra parte, se pudo dar respuesta a las inquietudes planteadas en la mesa redonda número cuatro en base a las problemáticas planteadas respecto a la transmisión de datos con MD5, señalando los protocolos y las formas correcta de hacerlo utilizando el algoritmo.

Se pudo cumplir con el objetivo de transmisión de archivos byte por byte donde el receptor, puede saber que para los primeros 16 byte de hash corresponde a un carácter del archivo enviado, pudiendo así de acuerdo con el menú de los números en hexadecimal encriptado, el mensaje a decodificar.

Finalmente, se puede concluir en base a los tiempos de encriptado que la matriz “enig” que contiene los HASH de los caracteres en hexadecimal reduce el tiempo, al no tener que encriptar el mensaje solucionando el problema de dificultad al buscar dentro de la matriz el hash correspondiente a la entrada sin encriptación alguna.

1 Bibliografía

[1] A. Humanidades, «concepto definicion,» 21 Diciembre 2016. [En línea]. Available: https://conceptodefinicion.es. [Último acceso: 17 Abril 2018].

[2] E. u. politecnica. [En línea]. Available: www.Icc.suma. [Último acceso: 24 Octubre 2018].

[3] V. P. L. Reference, «api.ning,» [En línea]. Available: www.api.ning.com. [Último acceso: 13 Diciembre 2018].

[4] cursos.aiu.edu, «cusos.aiu.edu,» [En línea]. Available: www.cursos.aiu.edu. [Último acceso: 20 Julio 2018].

[5] GMO Inernet Group, «GlobalSign,» 4 Mayo 2016. [En línea]. Available: www.globalsign.com. [Último acceso: 1 Junio 2018].

[6] Kaspersky, «latam.kaspersky,» Kasperskylab, 4 Abril 2014. [En línea]. Available: https://latam.Kaspersky.com. [Último acceso: 3 Abril 2018].

[7] IETF, «RFC HTML,» Marzo 2011. [En línea]. Available: https://tools.ietf.org/html/rfc6194. [Último acceso: 1 junio 2018].

[8] Rosettaacode, «rosettacode.org,» 28 Marzo 2018. [En línea]. Available: http://rosettacode.org. [Último acceso: 16 Abril 2018].

[9] M. t. Scripts, «TLM,» [En línea]. Available: https://www.movable-type.co.uk/Scripts/sha256.html. [Último acceso: 4 Junio 2018].

[10] U. d. Valencia, «UV,» [En línea]. Available: https://www.uv.es. [Último acceso: 1 Noviembre 2018].

[11] Copyright, «vpascal,» [En línea]. Available: www.vpascal.com. [Último acceso: 17 Julio 2018].

[12] z. service, «zeroshell,» [En línea]. Available: http://www.zeroshell.net. [Último acceso: 3 Noviembre 2018].

[13] C.alonso, «blog personal,» 20 Marzo 2015. [En línea]. Available: www.elladodelmal.com. [Último acceso: 1 Noviembre 2018].

[14] N. Cabanes, «Nachocabanes,» 2 Agosto 2014. [En línea]. Available: http://www.nachocabanes.com. [Último acceso: 14 Diciembre 2018].

[15] C. collaguazo, «In slideshare,» Universidad de Cuenca, 28 Junio 2017. [En línea]. Available: https://es.slideshare.net. [Último acceso: 10 Abril 2018].

[16] dit.upm, «Estudio de situacion del comercio electronico en España,» [En línea]. Available: www.dit.upm.es. [Último acceso: 6 Noviembre 2018].

[17] M. Figueroa, «Investigaciones y mas,» 21 Octubre 2015. [En línea]. Available: https://isla1md.wordpress.com. [Último acceso: 1 Junio 2018].

[18] P. Gutierrez, «genbetadev,» Genbeta:dev, 3 Enero 2013. [En línea]. Available: www.genbetadev.com. [Último acceso: 20 Marzo 2018].

[19] J. D. Muro, «Revista Sic,» [En línea]. Available: http://revistasic.com. [Último acceso: 1 Junio 2018].

[20] B. Ramos, «In slideshare,» 16 Octubre 2012. [En línea]. Available: https://es.slideshare.net. [Último acceso: 25 Marzo 2018].

[21] R. Rivest, «Ierf,» MIT Laboratory for Computer Science, Abril 1992. [En línea]. Available: http://www.ietf.org. [Último acceso: 17 Abril 2018].

[22] A. Vasquez, «In slideshare,» Universidad de cuenca , 19 Junio 2017. [En línea]. Available: https://es.slideshare.net. [Último acceso: 11 Abril 2018].