SIMULACIÓN DEL USO DE MÉTODOS MATEMÁTICOS EN EL DISEÑO DE MALWARE

TRABAJO FIN DE MÁSTER

Autor:
Aurelio José González Molina

Tutor:
Juan Antonio López Ramos

MÁSTER EN MATEMÁTICAS

SEPTIEMBRE, 2020
Universidad de Almería

Índice general

Abstract in English

Throughout this project, we will address the cryptography and its applications in the world of the computer security, these aimed at both strengthening and breaking it.

The project will be organized in five chapters. The first will be an introduction to put the reader in context. In the second, we will list a set of malware, pointing their purposes and giving some examples. Afterwards, their functioning and inner structures will be described. Those are based on three modules: an infectious, an attack and a defense module. In the third chapter, we will address the cryptography and the beginnings of its use for malicious purposes. We will also explain the existing cryptograpy types as well as the cryptosystems based on them, along with some examples. In the fourth chapter, helped by the Mathematica, we will use these cryptosystems to simulate the actions some malware do. Finally, in the fifth and last chapter, we will state the conclusions drawn out from the project.

Resumen en español

En este trabajo, vamos a tratar la criptografía y las aplicaciones que tiene en el mundo de la seguridad informática, tanto para reforzarla como para romperla.

Se va a estructurar en cinco capítulos. En el primero, se hará una introducción para poner en contexto al lector. En el segundo, se hará un listado de malware, viendo los objetivos de cada uno y algún ejemplo de ellos. Se continúa describiendo su funcionamiento y estructura internas. Se basan en un módulo de infección, en otro de ataque y en otro de defensa. En el tercer capítulo, se tratará la criptografía y los orígenes de su uso para fines maliciosos. También se describirá los tipos de criptografía que existen y los criptosistemas basados en ellas, exponiendo algunos ejemplos de ellos. En el cuarto, apoyándonos en Mathematica, usaremos estos criptosistemas para simular las acciones que llevan a cabo algunos malware. Y, para acabar, en el quinto y último capítulo, expondremos las conclusiones obtenidas del trabajo.

1 Introducción

En un mundo como el de hoy en día, en el que estamos en permanente contacto con ordenadores y teléfonos móviles y en el que la información que en ellos almacenamos y mediante ellos compartimos lo es todo, la capacidad de protegerlos es algo muy importante. Mucha gente sabe muy poco sobre cómo se protegen sus datos o sobre quién y cómo accede a ellos.

Detrás de la seguridad de nuestros dispositivos electrónicos se encuentra la criptografía. La criptografía combina conocimientos matemáticos e informáticos con los siguientes objetivos a la hora de intercambiar información:

- Confidencialidad: asegurar que solo tenga acceso a ella quien esté autorizado.
- Autenticación: asegurar que quien comparte la información es quien dice ser.
- Integridad: asegurar que la información recibida es la esperada, sin haber recibido alteraciones por alguien no autorizado.
- No repudio: impedir que quien comparte la información niegue haberlo hecho e, igualmente, que quien la recibe niegue haberla recibido.

Como veremos más adelante, hay dos tipos de criptografía, la simétrica o de clave privada y la asimétrica o de clave pública. La simétrica cifra y descifra la información haciendo uso de una sola clave. Esto da lugar a que, si a partir de la información cifrada se obtiene la clave, se pueda descifrar la información sin problema alguno. Entonces, para evitar esto, surge la criptografía asimétrica, la cual hace uso de una clave pública y de otra privada. La información solo se puede descifrar con esta última clave, pero obtenerla se hace casi imposible. Esto se debe a que la criptografía de clave pública se basa en complejos problemas matemáticos cuya solución es computacionalmente muy costosa.

Como hemos dichos antes, la criptografía es la encargada de la seguridad de nuestros dispositivos. Por ende, es la encargada de luchar contra software que tienen fines maliciosos como, por ejemplo, dificultarnos su uso, robarnos nuestra información o destruirla. Es decir, la criptografía es la base de la ciberseguridad.

Sin embargo, la criptografía no está limitada al ámbito de la seguridad. A finales de los años 90, Adam Young y Moti Yung pensaron en otra forma de usarla. Posicionándose en una situación totalmente distinta a la de quien recibe un ataque informático, pensaron en cómo podían aprovechar el gran potencial de la criptografía para diseñar software dañinos. Es en esto en lo que nos vamos a centrar en el presente trabajo, en ver cómo usar la criptografía con fines maliciosos, y, también, en realizar simulaciones con Mathematica de las acciones que lleva a cabo un software malicioso.

A lo largo del trabajo, veremos tanto un listado de software maliciosos como su estructura y funcionamiento. Para continuar, haremos una introducción a la criptología, constituida por la criptografía y el criptoanálisis, es decir, constituida por los procesos de cifrado y descifrado de información. Después, hablaremos sobre el trabajo que realizaron los anteriormente mencionados Adam Young y Moti Young, y trataremos con más detalle las criptografías de clave pública y privada, así como los criptosistemas que hacen uso de ellas. Por último, mediante estos criptosistemas, simularemos en Mathematica las acciones que llevan a cabo los software maliciosos.

2 Introducción al malware

2.1 Clasificación de malware

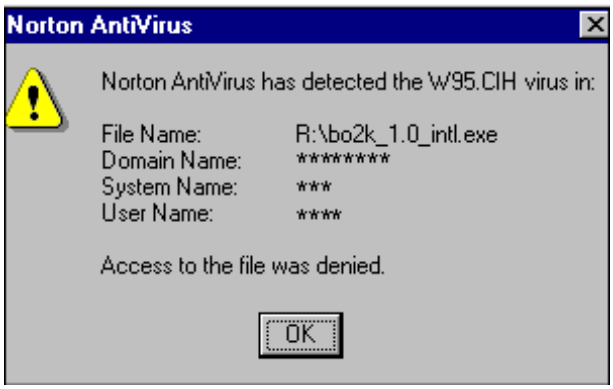
Ya hemos usado anteriormente las palabras «software» y «malicioso». Uniendo ambas, surge el término conocido como **malware**, que hace referencia a los programas cuyo objetivo es atacar a cualquier tipo de dispositivo digital (generalmente ordenadores, por lo que, de aquí en adelante, nos referiremos siempre a ellos). Conocer los malware sirve para saber cómo actuar contra ellos.

En este capítulo, vamos a introducir los tipos de malware existentes, viendo cómo infectan los sistemas y cómo se propagan. Aunque se suele tratar como iguales a los virus, a los gusanos y a los troyanos, la realidad es que son distintos, tal y como veremos a continuación.

Un **virus** es un tipo de malware adherido a algún archivo o programa que viaja de un ordenador a otro dejando copias suyas tras una acción determinada del usuario.

Cuando se descarga el archivo o programa en cuestión, el virus entra en el ordenador, pero, hasta que el usuario no realice una determinada acción como abrir o ejecutar el archivo, no estará activo. Los daños que puede provocar un virus van desde simplemente ralentizar el ordenador hasta bloquear algunos servicios o incluso dañar o eliminar archivos y otros software.

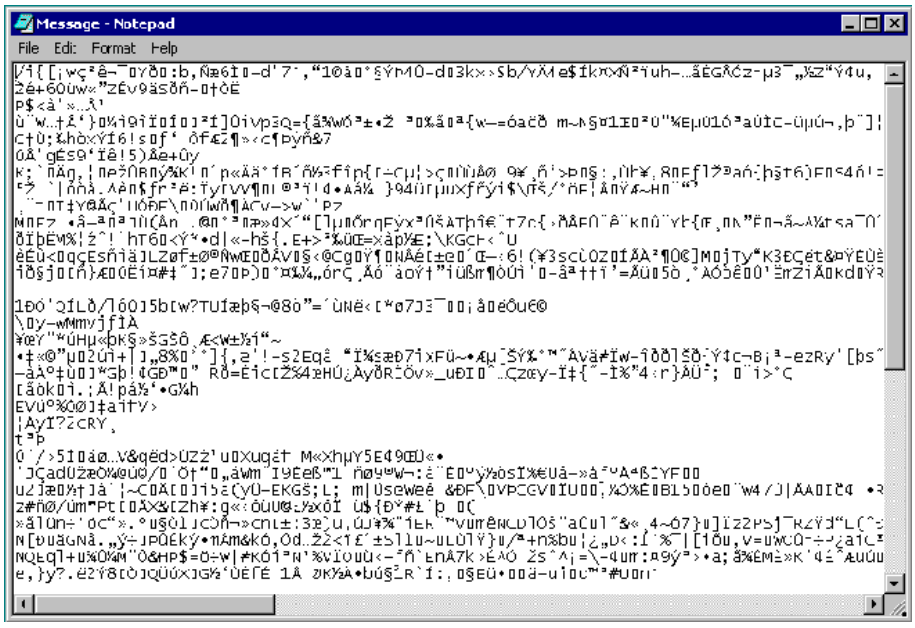
El virus ***Chernobyl*** (también conocido como ***CIH***) afectaba a ordenadores con *Windows 95* y *98*, infectando archivos ejecutables *.exe* y atacando a la BIOS, logrando que los ordenadores no arrancaran. A pesar de lograr un gran alcance, fue su propio creador quien creó y difundió un antivirus eficaz contra él.



Un **gusano** es un tipo de malware que se propaga de ordenador en ordenador dejando copias de sí mismo sin necesidad de que el usuario realice acción alguna.

Los gusanos pueden atacar a cualquier ordenador que esté conectado a internet. Una vez dentro, aprovecha las funciones de transporte de los archivos para viajar sin ayuda. Puede crear un número de copias tan grande que llegan a ocupar demasiada memoria, ralentizando los servidores de red, los web y los ordenadores particulares, pudiendo provocar que dejen de responder. Se suele encontrar gusanos en programas de descargas como Ares, eMule o BitTorrent y en enlaces de descarga directa como los de Megaupload.

Un **ejemplo** de gusano es aquel que se difunde por correo electrónico entre todos los contactos de la libreta de direcciones, haciendo después lo mismo con los contactos de cada una de las personas que reciban el primer mensaje. Un conocido gusano es el llamado ***Mydoom***, identificado en 2004. Este gusano contaba con una puerta trasera (lo cual se definirá más adelante), se propagaba a través del correo electrónico y a través de KaZaA, una aplicación destinada al intercambio de archivos. *Mydoom* tenía dos versiones, sirviendo la última para evitar o impedir la actualización de los antivirus. También realizó ataques de denegación de servicio (DDoS[1]) a los servidores de Microsoft. Tras entrar y ser ejecutado, abría un archivo de texto como el siguiente para instalar el código malicioso y difundirse entre los contactos.



Los gusanos más modernos tienen la capacidad de abrir túneles en los ordenadores, mediante los cuales los atacantes pueden controlarlos a distancia.

Los virus y los gusanos son similares por su diseño. Incluso se puede considerar a los gusanos como una subclase de los virus. La mayor diferencia entre ambos reside en que los gusanos son software independientes, es decir, no necesitan estar conectados a ningún tipo de archivo para expandirse, todo lo contrario a los virus. Además, los gusanos se expanden de forma más rápida y amplia que los virus.

Un **troyano** es un tipo de malware que se hace pasar por algún tipo de archivo o programa seguro para ser descargado y, entonces, realizar acciones distintas a las que pretende el usuario.

Una vez descargado, un troyano puede atacar al ordenador de diversas formas. Puede abrir ventanas para molestar al usuario, eliminar o robar archivos o datos, introducir otros malware e incluso crear puertas traseras a través de las cuales se les da acceso al ordenador a programas maliciosos o a ciberdelincuentes que pueden robar información confidencial y personal.

Por ejemplo, un conocido **troyano bancario** accedía a los teléfonos móviles haciéndose pasar por aplicaciones con otros fines. Entonces, realizaba acciones fastidiosas, como no parar

[1]Un ataque de denegación de servicio es un ataque que niega el acceso a algún recurso. Suele desconectar de internet a la víctima tras aumentar su consumo de ancho de banda.

de abrir ventanas, con el objetivo de obtener permisos como administrador. Ante esto, la víctima tenía dos opciones. La primera, y recomendada, era apagar el móvil y restablecer los valores de fábrica. La segunda era conceder los permisos a la aplicación, lo que causaría un gran daño a la víctima, pues el troyano intentaría obtener sus datos bancarios.

Un **ransomware** es un tipo de malware que secuestra información del ordenador de la víctima con el objetivo de chantajearla.

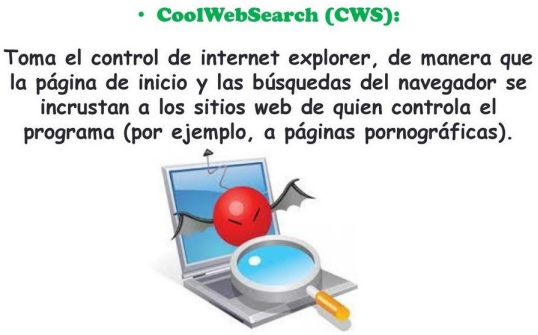
Este chantaje se puede basar en publicar dicha información o, simplemente, en negarle el acceso a ella a la víctima a menos que pague el rescate pedido. También pueden borrar la información o negar al usuario el acceso al ordenador.

Un curioso caso de ransomware fue el llamado ***Jigsaw***. Su nombre hace referencia a la saga de películas de terror *Saw*, pues borraba archivos de las víctimas de forma progresiva cada hora que pasaba sin que pagaran el rescate. Además, hacía uso de imágenes de la película.

Un **spyware** es un tipo de malware que, tras ser instalado en el ordenador y de forma secreta, reúne información personal de la víctima y la envía a otro ordenador.

Un spyware reúne información privada sobre la actividad del usuario en internet, obtiene direcciones de correo electrónico, contraseñas u otros tipos de información importante. Esto logra hacerlo mediante, por ejemplo, un registro de teclas pulsadas o haciendo un grabado de pantalla (**desktop recorders**).

El spyware conocido como ***CoolWebSearch*** aprovechaba las debilidades de la seguridad de Internet Explorer para cambiar su configuración, mostrar anuncios aleatorios y/o acceder a webs por su cuenta.



Un **adware** es un tipo de malware que toma el control del navegador y, entonces, genera falsa publicidad con el objetivo de reunir información o recaudar dinero. También puede llevar a entrar a webs comprometidas. Los anuncios que genera suelen aparecer como ventanas emergentes tras la instalación de programas gratuitos.

Uno de los primeros adware que cobraron importancia fue ***DollarRevenue***, el cual instalaba en el navegador una barra de herramientas para controlar las webs que visitaba la víctima y que mostraba anuncios falsos en dichas webs o en ventanas emergentes. Además, contribuyó enormemente al ataque de unos bots que afectó a una gran cantidad de ordenadores.

Publicidad en ventanas emergentes

Una **puerta trasera** o **backdoor** es un tipo de malware que aprovecha las vulnerabilidades en la seguridad de un ordenador para generar una vía de acceso no autorizada mediante la cual se puede tomar el control del ordenador sin ser detectado.

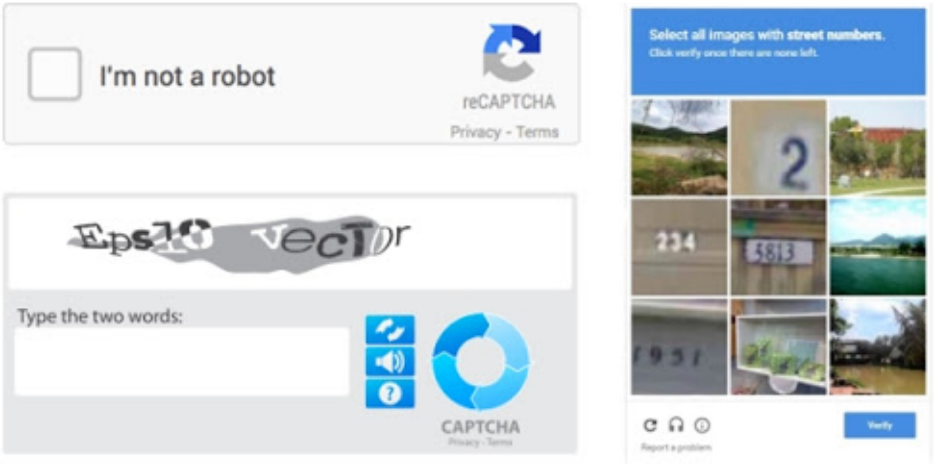
Una puerta trasera puede estar presente en un ordenador por diversos motivos, como haber sido creada por un error del usuario o por un troyano con esa capacidad, o el ordenador puede haber sido vendido con ella, por ejemplo.

Un **bot** es un tipo de malware que interactúa con servicios de red y que, tras infectar un ordenador, se conecta a un servidor central que controlará una red completa de ordenadores infectados.

Los bots se propagan como los gusanos, pueden realizar registros de teclas como los spyware y tienen otras habilidades como las de transmitir spam o abrir backdoors, entre otras. Para

combatirlos, al acceder a algún sitio web, se suele recurrir a las siguientes estrategias, con las que se pretende comprobar si se es bot o humano:

- Teclear una clave alfanumérica cuyos símbolos estén distorsionados y sean difíciles de leer.
- Marcar una casilla en la que se afirma no ser un robot.
- Mostrar unas imágenes de entre las cuales habrá que seleccionar aquellas en las que aparezca un determinado objeto, como semáforos, motocicletas o, como en el ejemplo de la siguiente imagen, calles numeradas.



Un **exploit** es un tipo de malware que ataca determinadas vulnerabilidades de la seguridad de un ordenador para llevar a cabo su verdadero objetivo. Este objetivo suele ser introducir en el ordenador otro tipo de malware. Aunque, a veces, simplemente se usa para descubrir brechas en la seguridad.

Existen los llamados **kits de exploit**, que son programas que almacenan los exploit más importantes. Estos kits se pueden encontrar en algunas webs y son usados por los hackers. Uno de los más usados es ***RIG***.

Un **keylogger** es un tipo de malware que reúne información del usuario por medio de un registro de teclas. Son similares a los spyware.

Un modo de contrarrestarlos es crear un teclado virtual que cambie la posición de los símbolos con cada ejecución y usarlo para escribir contraseñas. Esto es usado, por ejemplo, por numerosas entidades bancarias para el acceso a la banca online.

Un **riskware** es un tipo de malware que abre brechas en la seguridad de un ordenador, las cuales pueden ser aprovechadas para atacarlo.

Incluyen varios programas, algunos de ellos para administrar los dispositivos de forma remota, descargar archivos o monitorear las acciones realizadas.

Un **rootkit** es un tipo de malware que, evitando la seguridad del ordenador, oculta su presencia y la de otros malware permitiendo al atacante actuar de forma más cómoda.

Un ejemplo es el llamado ***Necurs***, el cual infectó en un mes más de 80000 usuarios de *Windows 7*. Generalmente, accedía a los ordenadores a través de páginas web y de correos electrónicos. Una vez dentro, descargaba una gran cantidad de malware y creaba backdoors que permitían al atacante controlar de forma remota el ordenador. En 2019, una evolución de este malware causó un gran daño a una importante empresa española.

Un **scareware** es un tipo de malware que plantea al usuario soluciones de pago a falsos problemas.
Por ejemplo, falsos antivirus que limpian falsos malware.

La mejor forma de proteger un ordenador de los malware es mantener el sistema operativo limpio descargando regularmente parches y mejoras y asegurándose de que el ordenador está protegido por la última versión de un antivirus de confianza.

2.2 Funcionamiento y partes de un malware

En esta sección, vamos a ver los distintos módulos que forman la estructura de un malware:

- Módulo de propagación o infección.
- Módulo de ataque o payload.
- Módulo de defensa.

1.2.1 Módulo de propagación o infección

El módulo de propagación o infección de un malware es el encargado de buscar víctimas a las que infectar. Esto se consigue haciendo uso de unos infectores que adjuntan al código del archivo atacado o archivo host el código malicioso del malware. Los primeros infectores fueron los virus. Aunque a día de hoy no se usan por los avances tecnológicos, es interesante estudiarlos. Es por eso que vamos a ver los tipos que hay:

- Virus del sector de arranque.
- Virus multiparte.
- Infectores de archivos.

1.2.1.1 Virus del sector de arranque

Infectando el sector de arranque del disco, el cual contiene el código ejecutable, el virus logra su principal objetivo: controlar el flujo de ejecución del sistema. Para ello, toma el control de las primeras instrucciones del código del sector de arranque, al que le devuelve el control tras haber sido ejecutado. El espacio de código del que dispone este sector es muy reducido, lo que obliga al virus a ocultar su código en otros sectores del disco.

1.2.1.2 Virus multiparte

Estos virus infectan primero a los archivos host y después al sector de arranque. Si, además, admiten infecciones MBR (Master Boot Record, la estructura de datos más importante en el disco), buscan discos duros fijos a los que intentan infectar.

1.2.1.3 Infectores de archivos

Los infectores de archivos se adhieren a los archivos host, los cuales, una vez infectados, propagan el virus. Al ejecutarse un archivo infectado, lo que se ejecuta primero es el código del virus. Es después cuando se ejecuta el código del archivo host, logrando, así, que la víctima no aprecie nada a simple vista. Los archivos se pueden infectar de dos formas:

- Mediante infección directa: el virus busca archivos en el sistema a los que infectar.
- Mediante infección en memoria: el virus espera a que el archivo host se ejecute y cargue en memoria para adherirse a su código. Una vez hecho esto, el código del virus se guarda en el archivo en el disco.

Por tanto, la infección directa tiene un mayor alcance y potencial, ya que puede infectar a todos los archivos del sistema, independientemente de si ya han sido ejecutados o no.

Hay varios tipos de infectores de archivos:

- Macros.
- Scripts.
- Ejecutables.

Macros

Un **macro** es un conjunto de instrucciones que lleva a cabo, de forma automática, una determinada tarea. Se pueden construir utilizando un lenguaje de programación de macros que permite al usuario programar tareas para que se ejecuten automáticamente. Este lenguaje también puede ser usado para la creación de virus. Los infectores creados a partir de este lenguaje se conocen como **macrovirus**.

Los principales macrovirus se han creado usando el lenguaje macro de Microsoft Office. Estos son:

- Macrovirus de plataforma única (Word, Excel, Access, PowerPoint): solo infectan archivos del mismo tipo. Por ejemplo, un archivo Excel solo puede infectar a otros archivos Excel.
- Macrovirus multiplataforma: a diferencia de los macrovirus de plataforma única, estos pueden infectar a archivos de Microsoft Office de otro tipo. Por ejemplo, un archivo Excel puede infectar a un archivo Word. Además de a los archivos de Microsoft Office, también pueden infectar a los archivos ejecutables.

Scripts

Los scripts pueden ser independientes al archivo host, no como los macros. Los lenguajes de script más usados son JavaScript y VBS. Cualquier archivo que use o pueda interpretar scripts puede ser infectado.

Ejecutables

Algunos infectores de archivos ejecutables solo pueden infectar archivos del mismo tipo, es decir, por ejemplo, archivos ejecutables del tipo *.exe* solo pueden infectar a otros archivos *.exe*. Sin embargo, hay otros que pueden infectar a cualquier tipo de archivo ejecutable. Sea como fuere, siempre se siguen los mismos patrones de infección, los cuales nos permiten clasificar los virus y los infectores de archivos. Dichos patrones son los siguientes:

- Virus que sobrescriben.
- Virus que acompañan.
- Virus que parasitan.

Veámoslos más detenidamente:

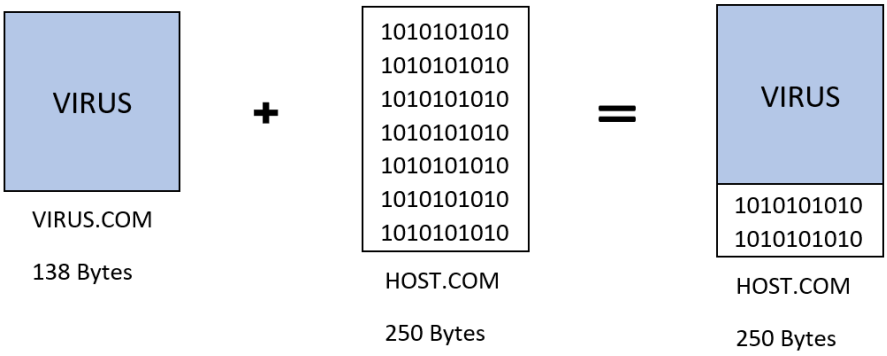
Virus que sobrescriben

Este es el tipo de infector más destructivo, pues sobrescribe el código del archivo host con el suyo. Los primeros ataques de estos virus se basaban en reemplazar el archivo host por una copia del virus y dotar a este último del nombre del host. Esto hacía más fácil su detección, pero también beneficiaba su tarea destructiva.

La única forma de recuperar un archivo sobrescrito es mediante una copia de seguridad previa a la infección.

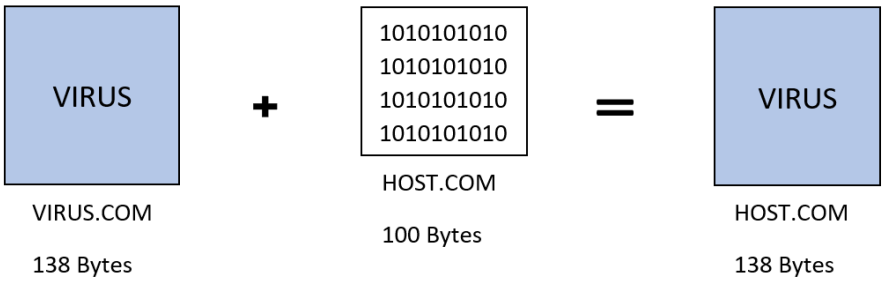
Podemos ver, a continuación, ejemplos de los resultados de ataques de estos virus:

- Si el archivo host posee un tamaño mayor que el del virus, se obtiene un archivo compuesto por el virus y por los bytes restantes del final del archivo host.



En este caso, el código del virus se ejecuta antes que el del archivo host.

- Si el archivo host posee un tamaño menor o igual que el del virus, se obtiene el mismo virus, pues ha sobrescrito el archivo host por completo. Así, solo se ejecutará el código del virus.



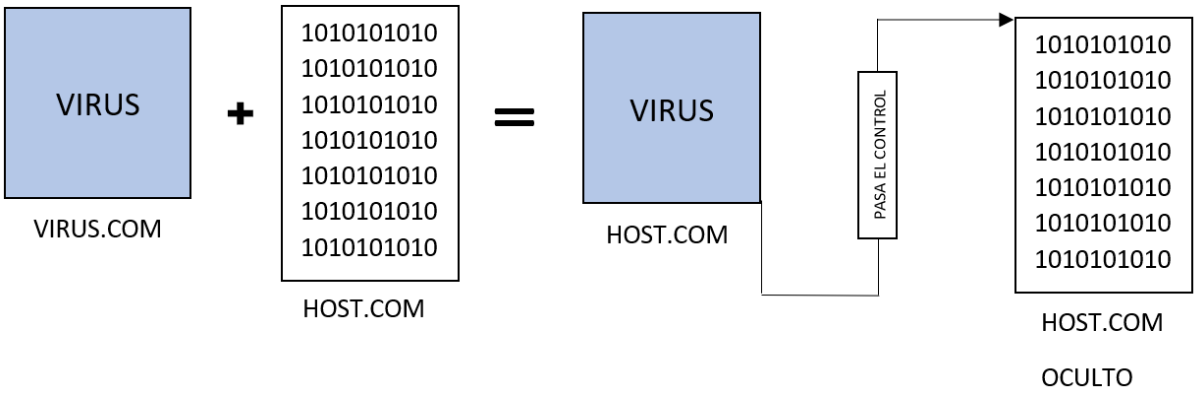
Virus que acompañan

Al contrario que los virus que sobrescriben, estos no adjuntan su código al del archivo host. Sí que tienen en común que el primero en ejecutarse es el código del virus. Después, le pasa el control al código del archivo host para que se ejecute como estaba previsto y, así, no levantar sospechas. Lo que hace que esto sea posible es lo siguiente:

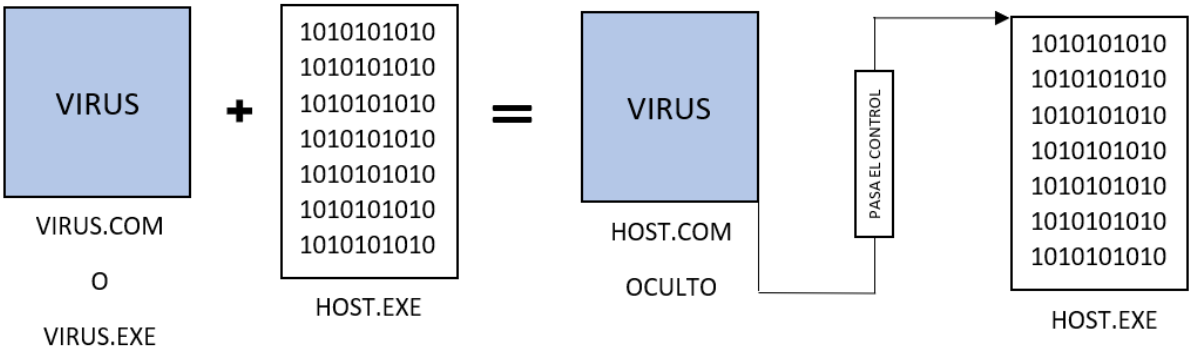
- Jerarquía de ejecución del tipo de archivo. El orden es *.com*, *.exe* y *.bat*.
- Posibilidad de ocultar un archivo o parte de él.

Para evitar estos problemas, es recomendable acostumbrarse a escribir el nombre completo de un archivo cuando se vaya a ejecutar, incluyendo su extensión.

Veamos algunos ejemplos:



Se puede observar que, cuando el virus infecta el archivo host, se apropia de su nombre y lo oculta. Así, al ejecutar el archivo llamado HOST.COM, se está ejecutando realmente el virus. Después, le pasa el control al verdadero archivo host para que se ejecute.

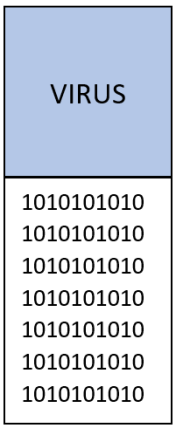


En este caso, un virus de extensión *.com* o *.exe* infecta un archivo host de extensión *.exe* y, en lugar de ocultar el host, se cambia el nombre y se oculta a sí mismo. Así, haciendo uso de la jerarquía de ejecución, logra ejecutarse primero. Después, le pasa el control al código del archivo host para que se ejecute.

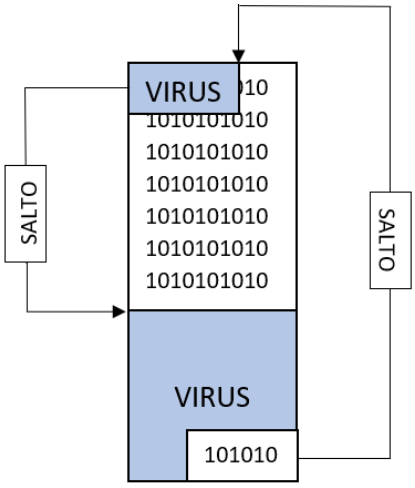
Virus que parasitan

Este tipo de virus se adhiere al archivo host a la vez que lo infecta, sin evitar que funcione como debe. Toma el control y guarda la ubicación de la primera instrucción del archivo host. Entonces, el virus se ejecuta primero y después pasa el control al código del archivo host. Hay dos tipos de virus que parasitan:

- Prepending virus: simplemente se adhieren al inicio del código del archivo host, logrando, así, ejecutarse antes que el propio host.



- Appending virus: el virus se adhiere al final del archivo host y secuestra su primera instrucción. Además, guarda su ubicación. De esta forma logra ejecutarse primero, pasándole después el control al código del archivo host.



1.2.2 Módulo de ataque o payload

El **payload** de un malware es el módulo de ataque, es decir, es la función que muestra qué tipo de malware es, pues contiene la parte maliciosa del código. Por tanto, puede ser considerado como la parte más importante del malware.

El objetivo principal de un malware es que el payload se ejecute correctamente. Sin embargo, a veces, puede encontrarse con problemas que le impidan ejecutarse por completo y que tienen que ver con las dependencias del malware. Estas dependencias pueden ser de varios tipos:

- Dependencia del entorno.
- Dependencia del programa.
- Dependencia del tiempo.
- Dependencia de acciones.
- Dependencia del usuario.
- Dependencia del archivo.

Dependencia del entorno

Los malware dependen de los entornos y solo tendrán éxito si se ejecutan en el adecuado, es decir, en el sistema operativo para el que están diseñados. Además, para facilitar su trabajo, se les puede dotar de la capacidad de desactivar el firewall de la red y la protección del antivirus, entre otras funciones presentes en el sistema. También hay que tener en cuenta que si el sistema se actualiza, el malware no funcionará, por lo que se deberá crear una nueva versión suya actualizada.

Dependencia del programa

Los malware con funciones específicas suelen depender de programas. Por ejemplo, los gusanos de correo masivo tienden a necesitar que el usuario utilice un navegador de internet para acceder a su correo web o que esté presente en el ordenador un cliente de correo electrónico como Outlook.

Dependencia del tiempo

Los malware se pueden programar para que ataquen de inmediato o para que permanezcan inactivos durante un tiempo determinado antes de atacar. Esto depende del payload del malware, el cual determina la fecha y hora de ejecución.

Dependencia de acciones

Los malware también se pueden programar para que su payload se ejecute tras una acción o una serie de acciones producidas en el sistema, tales como pulsar una combinación de teclas.

Un ejemplo es el presentado por un keylogger, el cual entra en acción cuando se entra en la banca online y se realizan unas pulsaciones de telcas, las cuales son registradas.

Dependencia del usuario

Algunos malware, aunque se intenta que no sea así, dependen de que el usuario realice alguna acción. Infectan sistemas en función del rol que desempeña cada usuario, pues, en una empresa por ejemplo, no todos tienen acceso a la misma información.

Dependencia del archivo

Los malware que dependen de archivos suelen ser los que roban información. Actúan buscando documentos con palabras o extensiones clave codificadas en su código, y, si los encuentran, envían copias a los atacantes.

1.2.3 Módulo de defensa

Una vez dentro del ordenador, el malware se encuentra expuesto ante los investigadores, quienes pueden estudiarlo para conocer sus mecanismos y tecnologías y, así, combatirlo. Esto no les interesa a los atacantes, por lo que dotan al malware de un módulo de defensa para protegerlo. Este módulo dispone de unos mecanismos de protección que pretenden evitar la detección de los productos de seguridad y el análisis de los investigadores, o, al menos, dificultarles el trabajo, consiguiendo, de esta manera, ganar un valioso tiempo en el que el malware intentará llevar a cabo su objetivo.

También existen malware que no requieren protección, pues su ataque es tan rápido que no puede ser contrarrestado a tiempo.

Estos mecanismos de protección presentan varias capas de dificultad variable. Todas estas capas se pueden romper, pero la cuestión reside en el tiempo empleado para ello. Cuanto más difícil sea romperlas, más tiempo se tardará en hacerlo y más daño hará el malware.

Todo malware tiene dos estados, uno estático (antes de ser ejecutado) y otro dinámico (después de ser ejecutado). Se necesita protección en ambos estados. En el primero, para que no se le detecte y se pueda activar, y, en el segundo, para alcanzar su propósito. Veamos los mecanismos de protección más usados en ambos estados.

1.2.3.1 Mecanismos de protección de un malware en estado estático

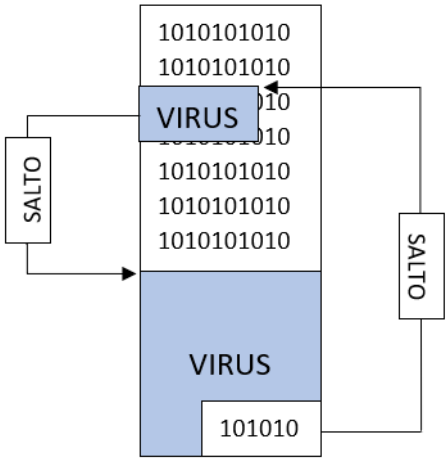
Un malware en este estado no está activo, por lo que un analista puede estudiarlo extrayendo su código. De esta forma, se puede descubrir cómo funciona y cómo detenerlo. Es por eso que los atacantes protegen el código del malware.

Los mecanismos más usados en este estado son:

- Ocultar el punto de entrada o inicio de ejecución.
- Encriptado básico del código.
- Polimorfismo.
- Metamorfismo.
- Mecanismos anti-Reversing.

Ocultar el punto de entrada o inicio de ejecución

El punto de entrada de un malware es un indicador de su primera línea de código. Por tanto, es importante ocultarlo del análisis y del escaneo de los antivirus. Una manera de ocultarlo cuando se trata con malware que infectan a otros archivos es introducir parte de su código en una parte intermedia del código del archivo host, tal y como aparece en la siguiente imagen.



Se puede observar que comienza ejecutándose el código del archivo host, pero, en pleno proceso, se ejecuta el código del malware. Al final, termina de ejecutarse el del archivo host.

Encriptado básico del código

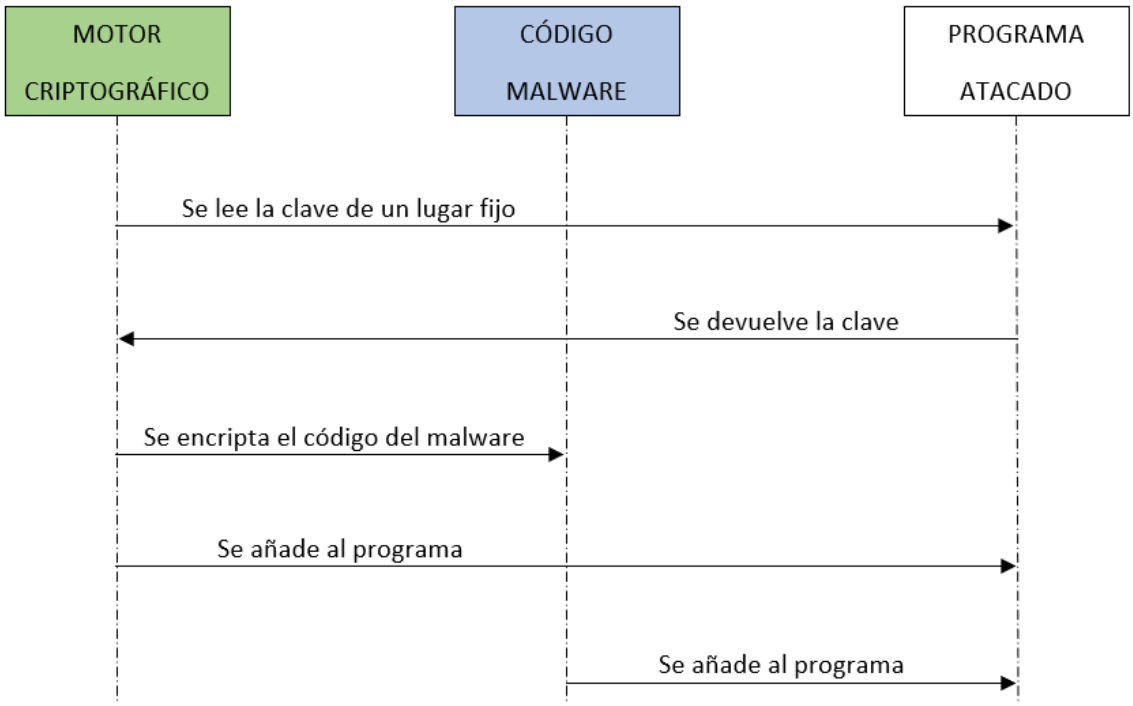
Este mecanismo consiste en ocultar el código del malware encriptándolo por completo. Un malware encriptado está formado por tres componentes:

- El motor criptográfico.
- La clave de encriptación y desencriptación.

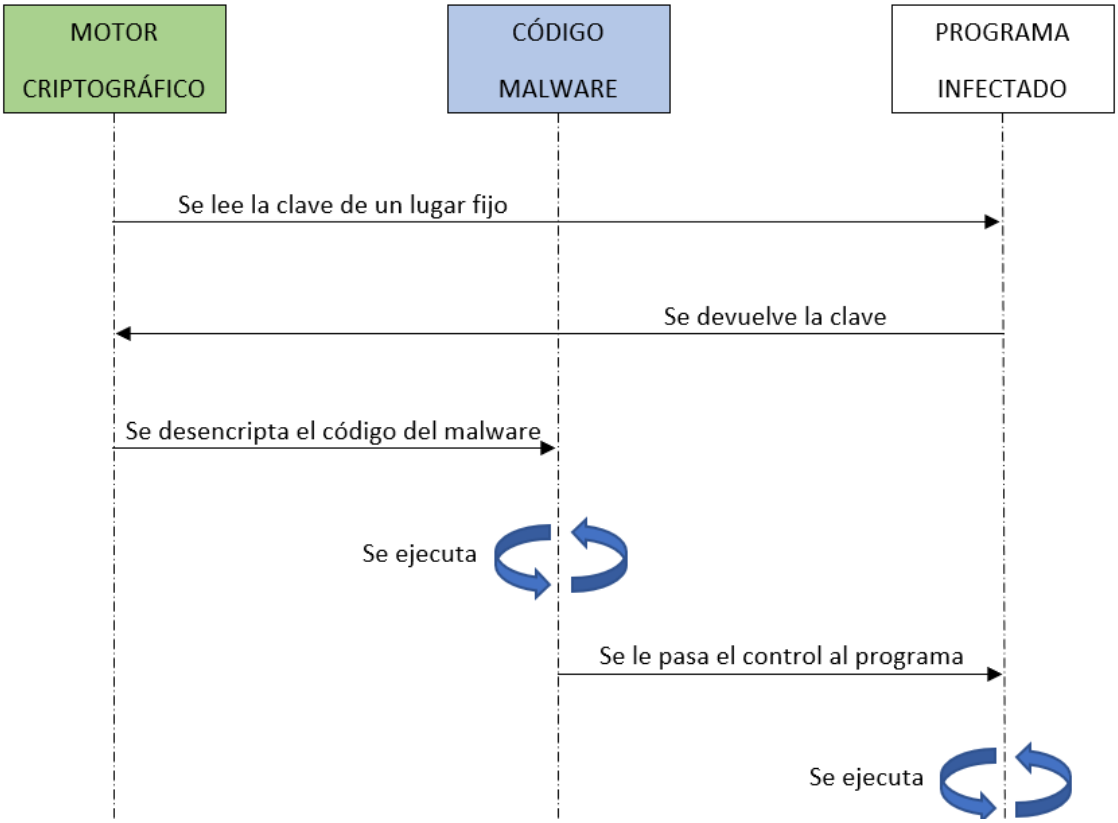
- El código del malware.

En primer lugar, se obtiene una clave a partir de una serie de bytes que están en una ubicación fija del programa que va a ser atacado. Como se ha dicho, esta ubicación es fija. Sin embargo, los bytes sitados ahí son diferentes para cada archivo, por lo que nunca hay dos infecciones iguales.

Después, se devuelve esa clave y se usa para encriptar el código del malware. Una vez encriptado, el motor criptográfico y el código del malware, por este orden, se añaden al programa objetivo.



Ya infectado el programa, el motor criptográfico vuelve a actuar como antes para obtener una clave, la cual será usada para desencriptar el código del malware. Después de desencriptarlo se ejecuta y, entonces, le pasa el control al programa infectado para que se ejecute también, tal y como se puede ver en el siguiente esquema.

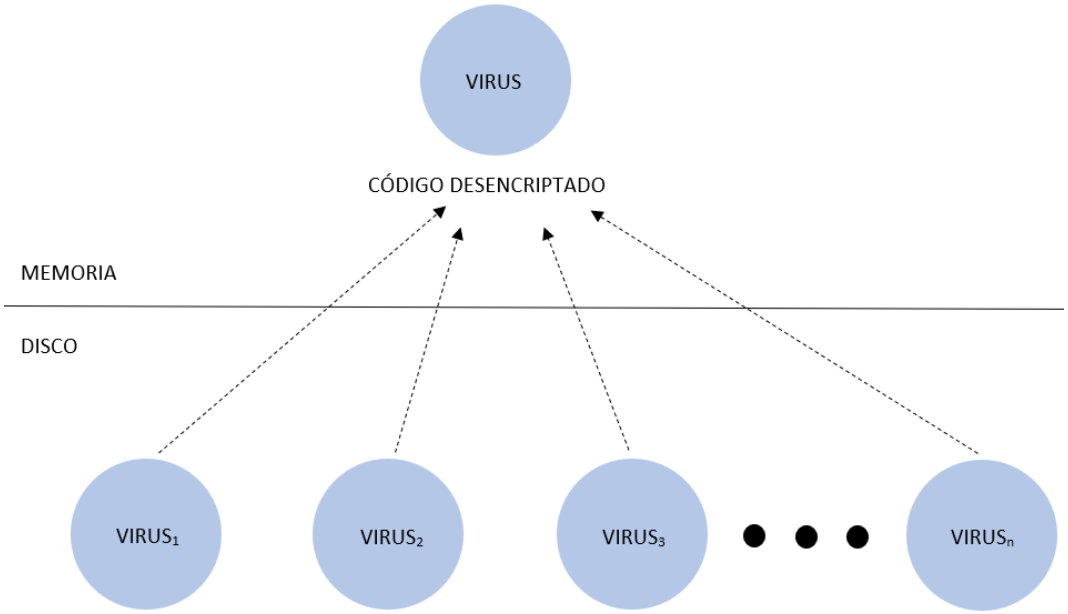


La debilidad de este mecanismo de defensa reside en que el motor criptográfico es constante. Esto permitió que los productos de seguridad hallaran la forma de detectar el mecanismo.

Polimorfismo

Para solucionar los problemas surgidos con el anterior mecanismo de defensa, se crearon unos motores de mutación, los cuales permitían que el motor criptográfico cambiara con cada infección sin cambiar su funcionalidad. Así, se conseguía que las tres componentes del encriptado del código del malware no se mantuviesen constantes. A este mecanismo se le llama **polimorfismo**. Un malware que usa este mecanismo es conocido como **malware polimórfico**.

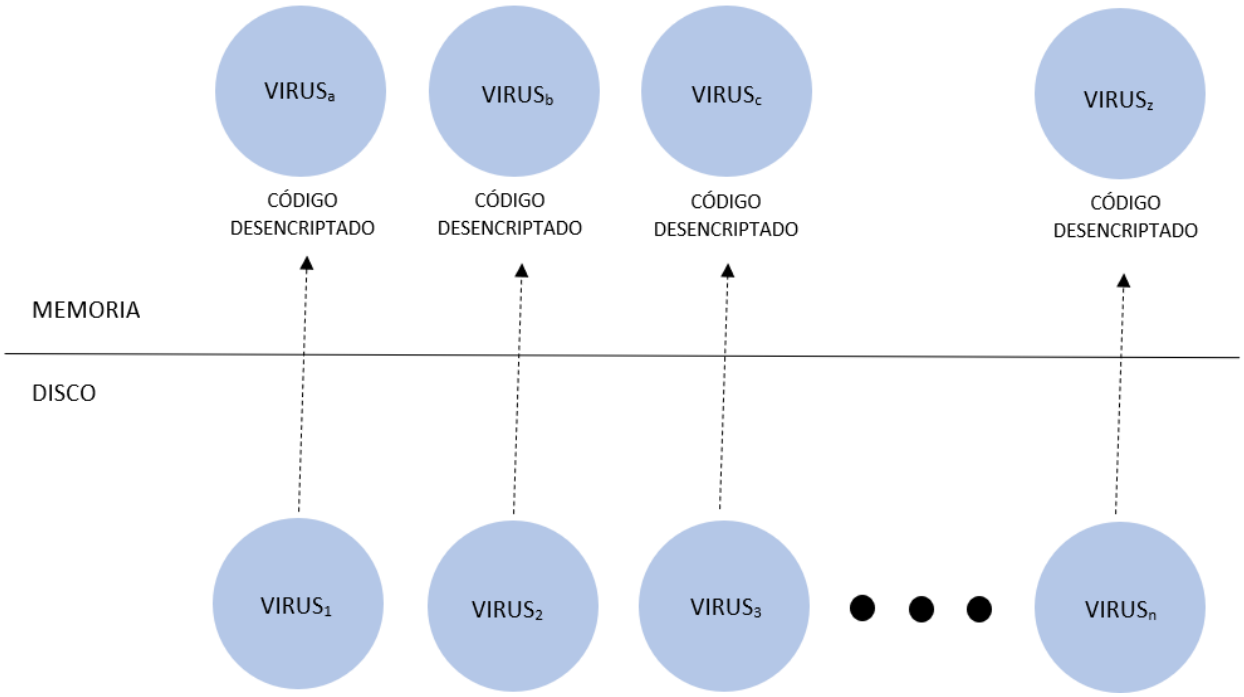
Sin embargo, sigue existiendo una debilidad. Tras el proceso de desencriptado del código, el malware está expuesto, pues el código es constante.



Entonces, surge la posibilidad de detectar el código del malware en la memoria. Pero, ante estos malware, no es efectivo un escaneo estático, por lo que los antivirus realizan escaneos dinámicos, analizando los programas que se están ejecutando en la memoria.

Metamorfismo

De nuevo, se intenta mitigar los problemas del mecanismo anterior. Los atacantes se dieron cuenta de que con el motor de mutación podían modificar todo el código del malware, obteniéndose cada vez una infección distinta en el disco y, también, en la memoria. A estos malware se les conoce como **malware metamórficos**.



Aún así, para transformarse, el malware debe analizar su propio código y modificarlo. Esto presenta una mínima debilidad y es que los investigadores también pueden modificar el código, aunque no es tarea sencilla.

Mecanismos anti-Reversing

Estos mecanismos se centran en revertir el proceso del ataque de un malware, ya que, de esta forma, pueden surgir ideas sobre cómo vencerlo. El mayor problema que presenta este proceso es el tiempo, pues, dependiendo de la dificultad del malware, se puede tardar demasiado tiempo en revertir el ataque. Si se tarda más en entender el malware que en que este actúe, se considera que el ataque ha tenido éxito. Quien realiza este trabajo recibe el nombre de **reversor**.

Entonces, los atacantes crean unas técnicas conocidas como técnicas anti-Reversing para defender al malware de este mecanismo y lograr retrasar y dificultar el trabajo de los reversores. Las más usadas son las anti-descompiladores y las anti-desmontadores.

El reversor necesita obtener el código del malware y, para esto, tiene que descompilarlo o desmontarlo. Para ello, puede aprovecharse de que las técnicas anteriores solo son efectivas con descompiladores y desmontadores compatibles con el malware. Es decir, al reversor le interesa usar descompiladores y desmontadores que no sean compatibles con el malware.

1.2.3.2 Mecanismos de protección de un malware en estado dinámico

Un malware se encuentra en este estado cuando se ejecuta. Por tanto, puede hacer uso de todas sus funciones y mecanismos de protección. Los más usados son los siguientes:

- Anti-debugging.
- Anti-sandboxing.
- Bloqueo del entorno.
- Antiescaneo de antivirus.

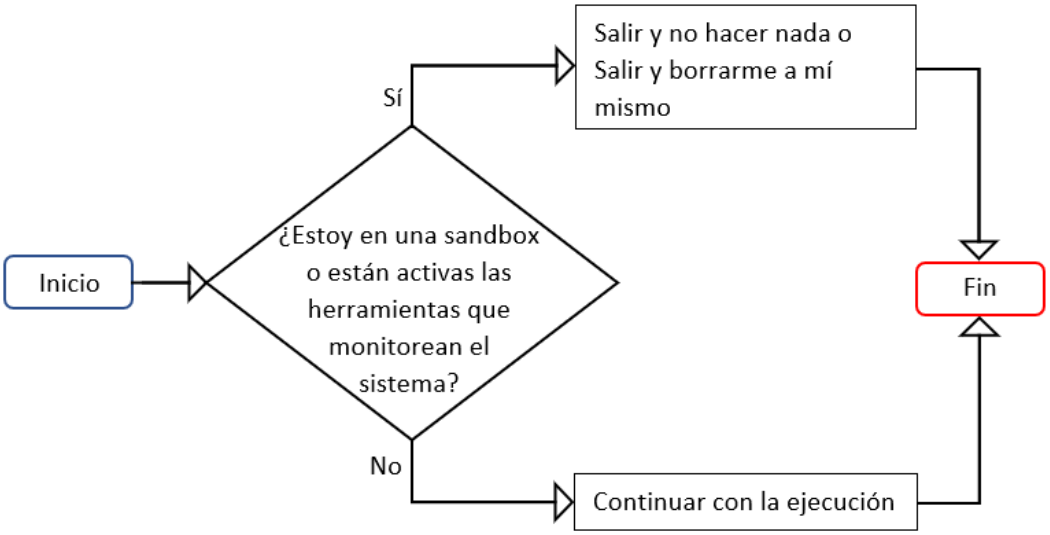
Anti-debugging

Los reversores utilizan los depuradores para observar todas las acciones que realiza el malware mientras se ejecuta. Por tanto, son una herramienta muy importante para ellos, lo que provoca que los atacantes intenten combatirlos. Entonces, los antidepuradores suelen engañar al depurador para que siga un flujo de ejecución sin salida.

Anti-sandboxing

Una sandbox es un espacio cerrado y limitado en el que se puede ejecutar un programa. Está compuesta por el entorno de la propia sandbox, que es donde se ejecuta el malware, y por unas herramientas que monitorean el sistema, las cuales registran todas las acciones realizadas por el malware.

El mecanismo anti-sandboxing pretende que el malware no se ejecute en la sandbox, evitando que se extraiga información de él. Para ello, cubre la detección de las dos componentes de la sandbox. Así, el malware no se ejecutará ni realizará ninguna acción si detecta que va a ejecutarse en una sandbox o que están activas las herramientas que monitorean el sistema.



Bloqueo del entorno

Cuando el malware se ejecuta por primera vez, usa información exclusiva del entorno en el que se encuentra para mutar y, así, no poder ser ejecutado en un entorno distinto. Es decir, se realiza un bloqueo del entorno, el cual impide que se realice un análisis dinámico.

Antiescaneo de antivirus

Si el antivirus posee una firma que coincida con las características del malware, puede detectarlo e identificarlo. En caso contrario, es decir, si el malware es nuevo, el antivirus utiliza una detección heurística. Entonces, si el malware detecta el antivirus, lo apaga. Otra forma de evitar el escaneo del antivirus es incluirse en su lista de no escanear.

3

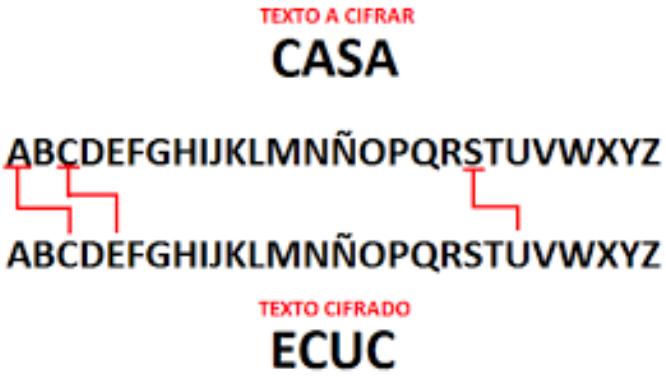
Criptografía

Han aparecido en el anterior capítulo palabras como «encriptado» o «criptográfico», las cuales proceden de la palabra *«criptografía»*. Pero, ¿qué es la criptografía? Este concepto surge de las palabras griegas *kryptós* (oculto) y *graphé* (grafo). La Real Academia Española la define como «el arte de escribir con clave secreta o de un modo enigmático».

Siempre ha existido la voluntad de enviar mensajes de forma que tan solo el destinatario pudiese leerlos. De esa necesidad surge la criptografía.

La **criptografía** consiste en la creación de procesos o técnicas para cifrar información. Sus orígenes se remontan a hace miles de años. Ya se usaba en tiempos de guerra para enviar mensajes que no pudiesen leer ni el mensajero ni los enemigos.

Una de las primeras técnicas es la conocida como cifrado de César, usada por Julio César. Esta técnica consiste en desplazar las letras del alfabeto y hacer corresponder a cada una otra letra. Por ejemplo:



Aquí, cada letra se desplaza dos lugares a la derecha, de forma que a la A le corresponde la C, a la B la D y, así, sucesivamente. Por tanto, la palabra CASA se cifraría como ECUC.

Nosotros nos vamos a centrar en cómo usar la criptografía para crear avanzados malware. Por ejemplo, los módulos de un malware vistos en el capítulo anterior hacen uso de técnicas criptográficas.

Sin embargo, la criptografía también tiene enemigos y son aquellos que intentan romper sus técnicas y descifrar la información. Es entonces cuando surge el término criptoanálisis.

El **criptoanálisis** consiste en la creación de procesos o técnicas para descifrar información previamente cifrada.

La criptografía y el criptoanálisis constituyen lo que se conoce como **criptología**.

A lo largo de este capítulo estudiaremos la función de la criptografía en el ámbito de los malware, así como las técnicas que usa.

3.1 Primeros criptovirus

Los pioneros en el uso de la criptografía en los malware fueron Adam Young y Moti Yung.

Adam era un estudiante que tenía interés por los virus. Cuando trató TEA (Tiny Encryption Algorithm), un algoritmo de cifrado simétrico, se dio cuenta de que era un algoritmo muy pequeño, es decir, ocupaba muy poco espacio en el disco y consumía muy poca memoria de acceso aleatorio, lo que hacía que fuese muy difícil de combatir. Esto llevó a Adam a investigar cómo usarlo para atacar a sistemas informáticos. La mayoría de virus usaban técnicas de encriptado en las que se escogía una clave que se usaba repetidas veces para ocultar la mayor parte del virus. El hecho de usar tantas veces la misma clave hacía que no fuese seguro. Es entonces cuando aparece la oportunidad de TEA, que permite crear virus polimórficos más difíciles de detectar y, por tanto, más devastadores. El virus almacena las claves de descifrado en el dispositivo infectado. Cuando el virus está en memoria, descifra los datos cuando se accede a ello, por lo que el usuario no nota que ha comenzado a cifrar.

Se modificó el virus One-half para que no contuviera el algoritmo de descifrado, dándole, así, al atacante la ventaja de que solo él podría descifrar los datos cifrados, quitándole esta posibilidad a la víctima. Sin embargo, esto tenía sus puntos débiles, ya que, a partir del algoritmo de encriptado, se podía deducir el de descifrado. Otra forma de ataque era la de borrar directamente los datos, sin cifrar nada.

El virus One-half permitía ver los puntos débiles de los virus. Con la máquina de Turing se podía saber en qué estado se encontraba un virus y se podía descubrir qué acciones iba a realizar. Fue entonces cuando se dieron cuenta de que la visión del virus del atacante y la de la víctima eran simétricas. Por tanto, había que hallar la forma de que el atacante pudiese seguir viendo la clave y que la víctima no lo hiciera, haciendo así sus visiones asimétricas. La criptografía asimétrica es la que se conoce como criptografía de clave pública.

En este nuevo ataque, se usa un criptosistema de clave pública como el RSA para la carga del virus. Antes de que el virus se despliegue, el atacante genera un par de claves RSA para montar el ataque. La clave pública se inserta en el virus, mientras que la privada es guardada en secreto. A partir de un hecho determinado, se cifran ciertos datos cruciales que serán secuestrados. Como la clave privada no se puede obtener a partir de la pública, a no ser que haya una copia de seguridad, la única persona capaz de descifrar estos datos es la que posee la clave privada, es decir, el atacante. De esta forma, se puede usar estos ataques para extorsionar a las víctimas.

Adam pensaba que podía hacer una tesis sobre esto. Se puso manos a la obra y consiguió ponerse en contacto con Moti Yung, profesor de su universidad. Moti ya estaba interesado en los ataques a sistemas informáticos y sabía del peligro que conllevaba trabajar con virus, pues se podía infectar por accidente algunos dispositivos durante la experimentación. Adam tenía un problema: a pesar de tener una sólida base de ingeniería, tenía unos conocimientos matemáticos demasiado básicos. Pero esto no impidió que Moti aceptara dirigir su tesis, pues poseía un gran y sincero interés en el tema. Enseguida, Moti notó que los conocimientos de Adam sobre virus eran más amplios de lo normal. Esto se debía a que se había formado por su cuenta. Entonces, Moti le ayudó para aprender teoría de números.

Adam abordaba la criptografía desde un punto de vista fuera de lo usual, incluso nuevo para Moti. Ambos bautizaron como **criptovirus** a los virus informáticos que contienen y usan una clave pública. Estos criptovirus son únicos, pues solo el atacante puede deshacer los cálculos realizados. Durante sus estudios encontraron dos defectos a la hora de realizar el ataque:

- Para poder cifrar grandes cantidades de datos tenían que dividirlas en bloques más pequeños que pudiesen ser cifrados por el RSA. Pero el RSA es más lento que otros algoritmos, por lo que el ataque llevaría más tiempo.
- Si el atacante le diese la clave privada a la víctima, esta podría hacerla pública para evitar que se pudiera extorsionar a más gente.

Se podía resolver estos defectos haciendo que el virus no cifrase los datos directamente con la clave pública, sino que generase una clave simétrica aleatoria que fuese la que cifrara los datos. Por tanto, la clave pública introducida en el virus cifraría la clave simétrica. Este método se conoce como cifrado híbrido. Si la víctima quiere recuperar los datos, debe enviar al atacante el texto cifrado de la clave simétrica. Entonces, el atacante descifra los datos con la clave privada y envía la clave simétrica a la víctima.

Estos ataques solo son exitosos si no existe copia de seguridad o si el virus no ha sido observado mientras realizaba el ataque. En este caso, que la víctima publicara la clave simétrica no serviría de nada, pues dicha clave es aleatoria para cada víctima.

Más adelante, crearon su primer criptovirus. Lo probaron en un ordenador Macintosh SE /30 previniendo los posibles problemas como que este virus se replicara en otro ordenador. El criptosistema híbrido del virus estaba basado en el RSA y en TEA. Para crear aleatoriamene las claves simétricas, se hizo uso del Truerand de ATT, un generador de números aleatorios. Truerand era muy importante, pues cualquier debilidad que presentase podría permitir a la víctima encontrar una forma de recuperar los datos. Truerand puede crear una cadena inmensa de números aleatorios, consiguiendo, así, que el criptovirus sea considerado como un algoritmo probabilístico con acceso a la aleatoridad generada por un dispositivo físico. Una vez usado este criptovirus, no se le devolvería la información a la víctima hasta comprobar que la información recibida es la correcta.

Continuando con la investigación se dieron cuenta de que los ataques de malware serían cada vez más poderosos conforme fuesen avanzando la criptografía y los algoritmos usados.

Desarrollaremos en las siguientes secciones los conceptos de criptografía de clave simétrica y asimétrica y de los criptosistemas.

3.2 Criptosistemas y criptografía de clave púbica y privada

Como ya se ha dicho antes, con el objetivo de proteger una información que se quiere transmitir, se realiza, en el ámbito criptográfico, una transformación del texto original en un texto cifrado. A esta transformación se la conoce como **cifrado** y al proceso inverso como **descifrado**.

Al conjunto de algoritmos que realizan estas transformaciones se le llama **criptosistema**. Los criptosistemas pueden ser de clave privada o de clave pública.

Definición 3.1. *Una **clave** es un parámetro que determina a un proceso de cifrado o de descifrado. Al conjunto de los posibles valores de una clave se le llama **espacio de claves**.*

Las claves que suele usarse son de tipo texto. Este texto se transforma en una cadena binaria que será usada como clave de cifrado. Entonces, el algoritmo de cifrado usa esta clave para cifrar el texto, el cual se puede descifrar, sin perder información, con el algoritmo de descifrado.

La diferencia entre un criptosistema de clave privada y uno de clave pública reside en la relación existente entre las claves de cifrado y descifrado.

Los primeros criptosistemas estaban compuestos por:

- Un algoritmo de cifrado E.
- Un algoritmo de descifrado D.
- Un espacio de mensaje M.
- Un espacio de texto cifrado C.
- Un espacio de claves K.

Si el emisor quiere enviar un mensaje $m \in M$ a un destinatario, primero lo cifra. El texto cifrado se denota por c y se representa como

$$c = E(m, k)$$

donde $k \in K$ es una clave aleatoria. Es decir, E calcula c a partir de m y de k. Posteriormente, D obtiene m descifrando c, para lo que hace uso de k. Se puede representar como

$$m = D(c, k)$$

A estos criptosistemas que usan la misma clave k en los algoritmos de cifrado y descifrado se les llama **criptosistema simétrico**. Sin embargo, el hecho de no variar la clave es lo que hace débiles a estos criptosistemas, pues se puede deducir la clave de descifrado a partir de la de cifrado. Se dice que estos criptosistemas son de **clave privada**.

Al darse cuenta de esta debilidad, se crea lo que se conoce como **criptosistema asimétrico**, en el que las claves de cifrado y descifrado son diferentes. A diferencia de lo que ocurre con los criptosistemas de clave privada, la clave de descifrado no se puede obtener a partir de la de cifrado. Incluso una de ellas se hace pública. A este tipo de criptosistema se le llama criptosistema de **clave pública** y está compuesto por:

- Un algoritmo de cifrado E.
- Un algoritmo de descifrado D.
- Un espacio de mensaje M.
- Un espacio de texto cifrado C.
- Un espacio de clave privada K.
- Un espacio de clave pública P.

En este caso, el emisor escoge aleatoriamente una clave privada $k \in K$, la cual usa para calcular la clave pública $p \in P$. Para cifrar el mensaje $m \in M$, se usa p y se calcula el mensaje cifrado c así:

$$c = E(m, p)$$

Una vez cifrado el mensaje, se le manda c al destinatario, quien, usando la clave privada k, descifra el mensaje, obteniendo m de nuevo. Para ello, calcula

$$m = D(c, k)$$

Este cálculo solo lo puede realizar el destinatario, pues es el único que conoce k.

Los criptosistemas de clave pública son tan seguros porque se basan en problemas que no son resolubles de forma computacional. En ellos, se hace uso de las llamadas **funciones de una vía**, pues permiten crear criptosistemas con un cifrado relativamente sencillo pero con un descifrado que, a menos que se conozca la clave, es irresoluble computacionalmente.

Definición 3.2. *Una **función de una vía** T_k, donde k es la clave de cifrado pública, es una transformación criptográfica de fácil aplicación pero con una inversa muy difícil (imposible computacionalmente hablando) de hallar sin conocer la clave de descifrado.*

Un ejemplo de función de una vía son las **funciones de hash**, las cuales resumen el mensaje que se desea transmitir para facilitar su difusión.

La siguiente sección se va a dedicar a ejemplos de criptosistemas de clave pública, pues son los que son realmente efectivos y los que se usan en la actualidad.

3.3 Ejemplos de criptosistemas de clave pública

2.3.1 RSA

El criptosistema de clave pública más conocido y usado es el llamado RSA. Le debe su nombre a las iniciales de los apellidos de quienes lo desarrollaron en 1978: Ronald Rivest, Adi Shamir y Leonard Adleman. Ellos se centraron en explotar la dificultad de factorizar números que sean producto de dos números primos muy grandes.

Antes de describir el algoritmo, introduzcamos un concepto que será necesario:

Definición 3.3. *Se define como **función de Euler** φ de un número n a la función que indica el número de números enteros positivos menores o iguales que n y coprimos*[1] *con él.*

Esta función viene definida por

$$\varphi(n) = n \prod_{p|n} \left(1 - \frac{1}{p}\right),$$

donde p es un número primo.
En particular, para un número primo p se tiene que

$$\varphi(p) = p - 1.$$

Además, para m y n coprimos, se tiene que

$$\varphi(mn) = \varphi(m)\varphi(n).$$

A continuación, vamos a describir los pasos a seguir del algoritmo del RSA:

[1] Dos números enteros son coprimos si su máximo común divisor es 1.

1. Se toma dos números primos muy grandes, p y q.
2. Se calcula $n = pq$.
3. Se calcula $\varphi(n) = \varphi(pq) = \varphi(p)\varphi(q) = (p-1)(q-1)$.
4. Se toma un número $e \in \mathbb{Z}$ tal que $1 < e < \varphi(n)$ y tal que $mcd(e, \varphi(n)) = 1$.
5. Se calcula el inverso d de e módulo $\varphi(n)$, es decir, se resuelve $d \cdot e \equiv 1 \ (mod \ \varphi(n))$.

La clave pública está formada por el par (e, n) y la clave privada por el par (d, n).

6. Cifrado del mensaje m: $c = m^e \ (mod \ n)$, donde c es el mensaje cifrado.
7. Descifrado del mensaje c: $m = c^d \ (mod \ n)$.

Veamos un ejemplo realizado en Mathematica:

```
In[1]:= p = RandomInteger[{2^1024, 2^1025}] (*Se crea un entero
   aleatorio p entre 2^1024 y 2^1025*)

Out[1]=
   33461956615367513775583994797863961270692903154538756438611870\
87275456574592255914504936564309321668542918874048394997972021349120\
40919244492875227106330590869389419760873094360910213723981464529012\
93500982272854446596428841050799011331700094795683589543747145205188\
04712998490684457575998015274602083247485 69

In[2]:= While[PrimeQ[p] == False, p++] (*Si p no es primo, se le va
   sumando una unidad hasta conseguir que lo sea*)

In[3]:= p

Out[3]=
   33461956615367513775583994797863961270692903154538756438611870\
87275456574592255914504936564309321668542918874048394997972021349120\
40919244492875227106330590869389419760873094360910213723981464529012\
93500982272854446596428841050799011331700094795683589543747145205188\
04712998490684457575998015274602083247489 01

In[4]:= q = RandomInteger[{2^1024, 2^1025}] (*Se crea un entero
   aleatorio q entre 2^1024 y 2^1025*)

Out[4]=
   34254514320429077419409006342611140381074781405478260880617162\
80478068635271301803108884567821300011466093138740143832373045106768\
33831818618553942950506805008330060364946632394076615777136270678600\
08491533095529883095309139918344432720364682867518226167789034005349\
59150573334695353973465617022704002846301 22

In[5]:= While[PrimeQ[q] == False, q++] (*Si q no es primo, se le va
   sumando una unidad hasta conseguir que lo sea*)
```

```
In[6]:= q

Out[6]=
   3425451432042907741940900634261114038107478140547826088061716 2\
80478068635271301803108884567821300011466093138740143832373045106768\
33831818618553942950506805008330060364946632394076615777136270678600\
08491533095529883095309139918344432720364682867518226167789034005349\
59150573334695353973465617022704002846302 13

In[7]:= n = p*q (*Se calcula el producto de p y q*)

Out[7]=
   1146223072070683002681437237205375213176862066240270168049347 3\
44369059785167545492405511985021956434118997893939124253840042595826\
09043806161731456864292028258039868855630733440796416429449368558481\
39462741924774988516572210725290917981227037141111557860025908673146\
33432852240420959370818618021331703651199722649723980833737886199492\
17840993892706389050954394750507330319307325329287368223337655317574\
54061650788745850190782098487492618822459265588801101371275957954238\
28283486134912827002743329123782343053397660295787089237615849768519\
82493175401084811205427676391034078548373498076588251362702926049328\
052863145913

In[8]:= fidn = (p - 1)*(q - 1) (*Se calcula la funcion de Euler de n
   *)

Out[8]=
   1146223072070683002681437237205375213176862066240270168049347 3\
44369059785167545492405511985021956434118997893939124253840042595826\
09043806161731456864292028258039868855630733440796416429449368558481\
39462741924774988516572210725290917981227037141111557860025908673146\
33432852240420959370818618021331703651199715878076887254078766700 19\
20643624287618871220535422457731503998253197280830101245157627320436\
14784485069862572230539379503682805993498415927765818436559221836646\
63348222793764527713444831794643026692319840875894865626844205167160\
54805265475330817918727056102924197317198714089405027020775656281959\
7444253766800

In[9]:= e = 65537 (*Se toma un entero mayor que 1 y menor que (p-1)(q
   -1) y que sea coprimo con este ultimo. Para asegurarnos de que son
    coprimos hemos tomado un numero primo*)

Out[9]= 65537

In[10]:= d = PowerMod[e, -1, fidn] (*Se calcula el inverso de e
   modulo fidn*)

Out[10]=
   1181254959605321116332823763688466696644113461920256452844239\
```

```
4317513205958499171850568118691484647994323377291705212643904432140 9\
4239104424701060865864211603910464736333567257669925380417933434920 7\
0637855499885454434046307751710659413835249414607118821103521655215 2\
6823329545190965652365089866969916728657404032898257925315543876425 6\
7893358536987928288195923574156866777609626538210880934583435539042 4\
6063359814484212167860337964323567933840247511728189232349985264578 9\
6093106134688582330234926742771548930831869101054273501132272787875 5\
8694398019750436306560912719824679626476038191553497299209893381270 9\
784373559073

In[11]:= m = 23 (*Mensaje a cifrar*)

Out[11]= 23

In[12]:= c = PowerMod[m, e, n] (*Cifrado del mensaje*)

Out[12]=
   7961749364742271781222532921078472276540136755152450895150945\
9650114434295479323492209798063758954406990804266762111778680092303 8\
0875115622785332399048020929363298497601739300266339998292991764513 3\
3756403613627415444050312452168204978431091036681853407066017668606 5\
5432704725069349369951545845440281895192697571994204565402470263505 3\
4543341677938835369192406410660347676871970368290841716541463952160 0\
7420323029631685879500611594219468921934714024104489556586952305096 4\
4446644839858944456795623949546454353443154610044651135774255730634 5\
9229909965114441867818227543278032789282494502443386392633007770966 2\
426696731065

In[13]:= PowerMod[c, d, n] (*Descifrado del mensaje*)

Out[13]= 23
```

2.3.2 ElGamal

Taher ElGamal desarrolló este criptosistema en 1984, el cual destaca por hacer uso de las claves pública y privada a la hora de cifrar el mensaje.

Al igual que el RSA, se basa en un problema computacionalmente irresoluble: el problema del logaritmo discreto. Veamos en qué consiste:

Consideremos un grupo cíclico G de orden n con una operación $*$ y con un generador g, es decir, un elemento $g \in G$ tal que $G =< 1, g, g^2, \ldots, g^{n-1} >$. Entonces, para cada $x \in G$ existe un número $k \in \mathbb{Z}$ tal que $x = g^k$, con $0 \leq k \leq n-1$.

La seguridad de este criptosistema reside en la dificultad de calcular ese k a partir de G, g y x. Describamos el algoritmo:

1. Se toma un generador $g \in G$.

2. Se toma aleatoriamente un número $a \in \mathbb{Z}$ tal que $1 < a < n-1$, el cual será la clave privada.

3. Se calcula $k = g^a$.

La clave pública será (g, n, k).

4. Cifrado del mensaje m. Para ello, hay que calcular lo siguiente:

   - Un número aleatorio $b \in \mathbb{Z}$ tal que $1 < b < n-1$
   - $h = g^b$
   - $d = k^b \cdot m = (g^a)^b \cdot m$

   Luego, el mensaje cifrado será (h, d).

5. Descifrado del mensaje (h, d): $m = h^{-a}d$. Veamos que esta expresión es cierta:

$$h^{-a}d = \left(g^b\right)^{-a}(g^a)^b\, m = g^{-ab}g^{ab}m = m$$

Este problema es computacionalmente intratable cuando se trabaja con grupos finitos grandes, como, por ejemplo, el grupo multiplicativo $\mathbb{Z}_p$, con p un primo muy grande. Se muestra, a continuación, un ejemplo realizado con Mathematica en el que, para calcular el generador g, se tiene en cuenta que g es generador si dada la factorización como producto de potencias de primos $p-1 = q_1^{e_1} \dots q_r^{e_r}$ se cumple que $g^{\frac{p-1}{q_i}}\ mod\ p \neq 1$ para todo $i = 1, \dots, r$.

```
(*Vamos a buscar un primo p tal que p-1=cq, con q otro primo*)

In[2]:= q =  RandomInteger[{2^1024, 2^1025}](*Se crea un entero
   grande que sera el primo q*)

Out[2]=
   24264213962693972294291383612409903740093512425456718469382737\
522400329099590747443244389280342758718716779290065784262186849381 53\
281602679615082873906152437342287525674803770244018528960427686263 46\
180302385169097349467474682134058539514189236481382912554328624930 04\
984585061982900459176983224811328680930384 9

In[3]:= While[! PrimeQ[q], q++](*Se le suma una unidad a q hasta
   conseguir que sea primo*)

In[4]:= q

Out[4]=
   24264213962693972294291383612409903740093512425456718469382737\
522400329099590747443244389280342758718716779290065784262186849381 53\
281602679615082873906152437342287525674803770244018528960427686263 46\
180302385169097349467474682134058539514189236481382912554328624930 04\
984585061982900459176983224811328680930414 7

In[5]:= PrimeQ[q](*Se comprueba que q es primo*)
```

```
22
23 Out[5]= True
24
25 In[6]:= p = 2*q + 1(*Se toma c=2 y se calcula p=2q+1, pues, de esta
      forma, se cumple la factorizacion p-1=cq*)
26
27 Out[6]=
      4852842792538794458858276722481980748018702485091343693876547 5\
28 0448006581991814948864887785606855174374335585801315685243736987 6\
29 5632053592301657478123048746845750513496075404880370579208553725 2\
30 3606047703381946989349493642681170790283784729627658251086572498 6009\
31 9691701239658009183539664496226573618608295
32
33 In[7]:= While[! PrimeQ[p], q = RandomInteger[{2^1024, 2^1025}];
34 While[! PrimeQ[q], q++]; p = 2*q + 1](*Si el p obtenido no es primo,
      se le suma una unidad a q hasta que tanto p como q sean primos*)
35
36 In[8]:= p
37
38 Out[8]=
      6425951992507640632177605379903806210475992127306376895988364 1\
39 5695543731458732903919162077154214337482801502724091175156242105292 9\
40 7256108006731173403206542343062587791027254147944661437290846244114 4\
41 6576787982339465830485390267894172427222905600971553830944319649534 7\
42 9595351049068161828155181544376177511317363
43
44 In[9]:= PrimeQ[p](*Se comprueba que p es primo*)
45
46 Out[9]= True
47
48 In[10]:= FactorInteger[p - 1](*Se comprueba la factorizacion de p-1*)
49
50 Out[10]= {{2, 1},
      {3212975996253820316088802689951903105237996063653188447994182 07\
51 8477718657293664519595810385771071687414007513620455875781210526464 8\
52 6280540033655867016032711715312938955136270739723307186454231220572 3\
53 2883939911697329152426951339470862136114528004857769154721598247673 9\
54 7976755245340809140775907721880887556586 81, 1}}
55
56 (*A continuacion, se comprueba si 5, por ejemplo, es generador*)
57
58 In[11]:= PowerMod[5, (p - 1)/2, p]
59
60 Out[11]=
      6425951992507640632177605379903806210475992127306376895988364 1\
61 5695543731458732903919162077154214337482801502724091175156242105292 9\
62 7256108006731173403206542343062587791027254147944661437290846244114 4\
63 6576787982339465830485390267894172427222905600971553830944319649534 7\
64 7959535104906816182815518154437617751131736 2
65
```

```
In[12]:= PowerMod[5, 2, p]

Out[12]= 25

In[13]:= g = 5(*Como cumple las condiciones de generador, se toma 5
   como tal*)

Out[13]= 5

In[14]:= m = 7686 (*Mnesaje a cifrar*)

Out[14]= 7686

In[15]:= a = RandomInteger[{2, p - 2}](*Se calcula a*)

Out[15]=
   59852048945854050207896470544426218619120779843927549535229 84\
067453692552206546913392890635455450761637344664709546229916901291 36\
233687869012931274582796839220179517289125430444988417853370795894 04\
185326262207960896535558726269380562220501625634553721534244854281 35\
825770734343280973165857338853430209018243 14

In[16]:= k = PowerMod[g, a, p](*Se calcula k*)

Out[16]=
   38110940223092844469541026573243615208458685613468973565428 68\
401751245400698283318128000229078694667839360473336101414256020563 04\
907474971619954194070786137981503479272554521168494598173541572589 43\
644283790869997165806774599635338553438031999137548103644572144717 01\
409515492392954072926846594492826835708483 98

In[17]:= b = RandomInteger[{2, p - 2}](*Se calcula b*)

Out[17]=
   40749901276872448753401646251602528904585978192993519901024 45\
230572806104514085209624908149683906485057941122542838265864377926 33\
700492909294672460460429373267704591137195196922316436001888623848 60\
014291076713739271650862011309298830248726429122144861528124680247 11\
631378434776815539906799904444649451076720 36

In[18]:= h = PowerMod[g, b, p](*Se calcula h*)

Out[18]=
   20564002022846802911684598477354454900157166023518121861985 99\
576194741268378599720892353508606163225866821553176136462888815708 53\
963408901638065191751861793254800988074619210832260482097763732501 56\
211737684102041709785622841309450326715618922517024926871240472630 19\
335784052228879030132555706102097701753588 22

In[19]:= d = Mod[m*PowerMod[k, b, p], p](*Se calcula d*)
```

```
111
112 Out[19]=
       24888245376843746460486333258906231049423136471673261911897 51\
113 940674520208918451244460233005914414041072994921372754443224512902 44\
114 453634278684490397659086070811140062076160014134491447227755000918 22\
115 116270998205543235494210970098458084697169827424625148951659091760 12\
116 648572220708475585473576341399981287002686 29
117
118 In[20]:= Mod[PowerMod[h, -a, p]*d, p](*Se descifra el mensaje*)
119
120 Out[20]= 7686
```

2.3.3 Paillier

Pascal Paillier, en 1999, desarrolló un nuevo criptosistema de clave pública basado en lo mismo que el RSA, es decir, en la dificultad que presenta la factorización de enteros grandes como producto de primos. Sin embargo, se diferencian en que en el criptosistema de Paillier, antes de cifrar, se trabaja en $\mathbb{Z}_n$, mientras que con el texto cifrado se trabaja en $\mathbb{Z}^*_{n^2}$.

Su algoritmo sigue los siguientes pasos:

1. Se toma dos primos muy grandes, p y q.
2. Se calcula $n = pq$.
3. Se calcula $\varphi(n) = \varphi(pq) = \varphi(p)\varphi(q) = (p-1)(q-1)$.

La clave pública es n y la clave privada está formada por el par $(n, \varphi(n))$.

4. Cifrado del mensaje $m \in \mathbb{Z}_n$:
   - Se toma un $r \in \mathbb{Z}^*_n$ aleatorio.
   - Se calcula $c = (1+n)^m r^n \left(mod\ n^2\right)$.
5. Descifrado del mensaje $c \in \mathbb{Z}^*_{n^2}$:

$$m = \frac{c^{\varphi(n)} \left(mod\ n^2\right) - 1}{n} \cdot \varphi(n)^{-1} \ (mod\ n)$$

donde $\dfrac{c^{\varphi(n)} \left(mod\ n^2\right) - 1}{n}$ se calcula en $\mathbb{Z}$.

Veamos, a continuación, un ejemplo realizado con Mathematica.

Nota. *Para que Mathematica pueda realizar los cálculos, tomaremos* $c = a \cdot b \ (mod\ n^2)$*, donde* $a = (1+n)^m \ (mod\ n^2)$ *y* $b = r^n \ (mod\ n^2)$.

```
In[1]:= p = RandomInteger[{2^1024, 2^1025}](*Se crea un entero
   aleatorio p entre 2^1024 y 2^1025*)

Out[1]=
   209579092998345769735849525224519916144015134014213874335665 29\
64985699016763601648607666439146567806943076810349573078026888855586\
13175587639855412683305939490771616615522414153849864337583582965615\
50767055276702115880980370591588908012082800417766317572744098625563\
06573141538985115082129183540704725557721 38

In[2]:= While[PrimeQ[p] == False, p++](*Si p no es primo, se le va
   sumando una unidad hasta conseguir que lo sea*)

In[3]:= p

Out[3]=
   209579092998345769735849525224519916144015134014213874335665 29\
64985699016763601648607666439146567806943076810349573078026888855586\
13175587639855412683305939490771616615522414153849864337583582965615\
50767055276702115880980370591588908012082800417766317572744098625563\
06573141538985115082129183540704725557722 53

In[4]:= q = RandomInteger[{2^1024, 2^1025}](*Se crea un entero
   aleatorio q entre 2^1024 y 2^1025*)

Out[4]=
   286023909015582253043824755642610829935287278619530363654284 76\
45013285586184239783570801841503294618842246880164045580233058590392\
11061694294929021366559037286149223762638699861973504541451205131889\
56339774378890548005837014830275918660063965309660517998506472502597\
95752423327276610513098053028582842213680 07

In[5]:= While[PrimeQ[q] == False, q++](*Si q no es primo, se le va
   sumando una unidad hasta conseguir que lo sea*)

In[6]:= q

Out[6]=
   286023909015582253043824755642610829935287278619530363654284 76\
45013285586184239783570801841503294618842246880164045580233058590392\
11061694294929021366559037286149223762638699861973504541451205131889\
56339774378890548005837014830275918660063965309660517998506472502597\
95752423327276610513098053028582842213680 07

In[7]:= n = p*q(*Se calcula el producto de p y q*)

Out[7]=
   599446314273271020532570835444675531885535171471997070736351 16\
90657819863264492992566955596344309978044044709014123354378527281528\
```

```
0362912172242113669391941094021498117312202501045443465336534503694\
6187015758236482068161052234712535263147181181150214998909086086774\
5347911902565814546110798887147111979498837474138294227171869619323\
2690644142240790557188152650077266102567801029949308869261589940370\
0650081352415083532858067197734869215822467055537140440083278368913\
4899551656057046861703875611450237163098066080436278008688129723502\
4202765200902426356103837561096199183653768101637772094545962571278\
9685455861571

In[8]:= fidn = (p - 1)*(q - 1)(*Se calcula la funcion de Euler de n*)

Out[8]=
   5994463142732710205325708354446755318855351714719970707363511\
6906578198632644929925669555963443099780440447090141233543785272\
8150362912172242113669391941094021498117312202501045443465336534\
5036946187015758236482068161052234712535263147181181150214998909\
0860867745347911902565814546110798887147111979498836978535292213\
2438468396489806525756356787655567091393019239408937787423237580\
0942361892960995846286011442815513839945645279366987707971427412\
5909592507806853448013123380805772009798622585361342028043603908\
6367551442536771510368950735334709681286787207121

In[9]:= m = 476981 (*Mensaje a cifrar*)

Out[9]= 476981

In[10]:= r = RandomInteger[{1, n}](*Se calcula r*)

Out[10]=
   2574048340222374967839706557101868062271895848480053371719212\
5313133153972709632961387792637506077909353737369543000680921596403\
9304502800640496583209567388480020470785012206028212747130181371590\
0629055872007214470291893682852096962341972485902559282004752230984\
0860916941317459959279149152882482842274021261640703004019242847332\
3778346397763707507400088520725168306513607391339818500664354721618\
5027626465683101119575708372835499252478916399669909621667116346563\
9579392194077239452061578959663695343892189071873952138494100623430\
8918158983673365753809626317373544601853598592879893276318556181164\
3128730606839919941

In[12]:= a = PowerMod[1 + n, m, n^2]

Out[12]=
   2859245024283790846446461696612367798742944516238846347968955\
1697216575761997611320875790472999053176374264893502655716948243212\
7052427822108282156201402374549674681934936916611511566695396855641\
0647835134896337939545335352685596541978235120562696621070039465479\
0755804379236719774478801847496399031661909333499725195771877066554\
2894430695320336101545207581622391865044628688923030662512937692624\
313476
```

```
5468962453556298958586173750042775653432216158617160784251362199682 7\
2535230484577462691423762930251455722916786571125763208620728036459 1\
8300791482916402257607645437292261828183579628872861714286277732112 1\
384321622307997152

In[13]:= b = PowerMod[r, n, n^2]

Out[13]=
   3542078952127361926348225229659397500835098230165920050139513\
4068099104718672185023498725961669047516196074419946283959614355639 7\
6375394078069782637511110056278669118284592452061391639506979411044 6\
9087912212106697524306491048775273935147175247338168189298335426728 2\
8311764376759578460573566525304447660673160791648118174885155470841 6\
0779045011515836860867250388982164482329884465383929833555231641543 0\
8782025750760949715750259596216069041518076701887232401307158378633 2\
2812813432635553187411288508054659840919679122638366110193450868877 6\
9552654121695185113517099526347225752989958763083248025906096696771 8\
3069535032548770518702444701310562755790555303996225388079508763649 2\
5014651227209721996516109940068054580329444934249664928523720235607 3\
7804472472754028759922597158072748773353869673252889091597442776112 1\
8138136541707280513248049190708378627798917010318776365450893748363 0\
7983262090452554864424587706480864670681457879650433816559752999662 9\
8299542765061550730686632067856636876588802106329962057210931746744 6\
5599642669401019939102423607922107635327842487625474227488763930284 8\
4253545377428692718790424633647999255205840197294558606502733186949 9\
1887066112824342064112772983630229546931837781718322481572597752542 8\
45916600185652853

In[14]:= c = Mod[a*b, n^2](*Cifrado del mensaje*)

Out[14]=
   1348724135650893803028568930944350719922462373719500260681806\
0795231915272948353820000624750955330066195654203361161433091478590 0\
8304674854282355772442758730544085742783417905809414126440211388509 5\
6463891803181041564326284713556948239516762569143988860482327467943 8\
6389227785298522034745124380190698799177057853413807279732949841755 5\
7510351275259946198772198751662329155154245560253063101211017881351 5\
7004349371200694543457233556852953265447591131204898990906790313178 5\
5943706325801251999533907053705658074216754257555652130015768867644 6\
0241980521367250750910594257079839536102796571724405161810828921362 0\
5330405264922743941720086271012606718609033253701320674870906999807 1\
8700525907505995846852056400930567056532033506505172854309969265181 3\
9884172845249329379975099172915024692975102276073299478269577836894 7\
4002456274111865340461337916951820345877442386247291008504127849977 1\
3257707502094724389695417090573084372237531311574636173486722027130 8\
0858361821151335287154737786086500029687421798427142370054351948269 5\
3022255349084121941782171713518254050899217564050142848357270562090 6\
5512752497998369214027285254153084502381940844757088740705312395265 7\
6764173635807503065882232021786575689689438611334806893121768037330 8\
10799607939805925
```

```
136
137 In[15]:= m = Mod[(PowerMod[c, fidn, n^2] - 1)/n*PowerMod[fidn, -1, n
       ], n](*Descifrado del mensaje*)
138
139 Out[15]= 476981
```

4 Criptografía y criptovirología

La **criptovirología** es la encargada del estudio de la criptografía para su posterior aplicación en la creación de malware.

En este capítulo, vamos a poner en práctica lo visto en los anteriores capítulos y a cifrar información como lo hacen los malware haciendo uso de criptosistemas de clave pública.

4.1 Un ejemplo de módulo de ataque: Ransomware

Vamos a ver cómo actuaría un ransomware que se vale del RSA. Para ello, vamos a trabajar con la siguiente información en binario, la cual, por su longitud de 8192 bits, que es exactamente un KByte, podría corresponderse perfectamente con un fichero de texto o una foto almacenada en un ordenador.

```
Cadena = \
1100011100001011001001100110000101000000101111100000000010000101100\
1000011111010010010111010001100011011001100001011101111000111001111\
0000100101110001111011010010011100111111001001100000011011011000110\
1101001111000010001000101101011110001110101100010110010001011110101\
1001010111101111000001100100010111010011100110001010110000110101110\
1101111110001001101110010101001010000111010100101001101111101111110\
0110000011110001000101001010111011010101111100000000000110001100110\
1111000001001010101110000101111101010000101110110000000110110110011\
0100000110001000001110110011100111101010100000011100110000011011010\
1110001010011001010111101001110011101011000011110000000100111010110\
1011100111100001110001000000010110000010100011011101010010100100101\
0101111100000111001011111110010111110001110100000100110111001100110\
1011000111100111100011011011000011110100000100101010101000011110111\
1010000100010100111100010000101000101000000000000100000100101101100\
0000011101001001101000011110111110011011100110011000101001001101000\
1001010110100110000011001101010000100011011110110110100010111010001\
1100001110010011110110111001111011001000110100001000111001001101001\
1011111100001110001100001000111000010100010011001000101101001010110\
1011011011011001010111001001001000110111000011011011101010110001111\
0101010101100001100110011010100011110011011010010110000001110010111\
0000011001011111001111101000110101100100100101001110011010100100110\
1001011011010110100110010010100110010010111110011010011001011010101\
0001011111110010101101010110001101101001111010101100110010110010001\
0111100110100001001011101011000110011001100000011111011100010110101\
0101011001100100111000110111001010110110001101011000010100111100000\
0011110011000101000010001100101000100111111111010111000001001010110\
1010001010111000101010100010001111010011101100001001100110001101001\
0110000000001101100010000001010101011101000001000110001010110010001\
0110101101011010010110110000011110100111100100011100110110101001000\
1001001111111110001111111011011111000011000001110111000001000111001\
1011001110101000001100001000111111001011000111001110010011000101001\
0110100010111011011111100011010101110100100001000100111111000010000\
0010101100001100110000010010111011011000001011101001011000111111101\
```

```
01011001110010001111001010000100101011001100100010011101110000111000\
10111010110001010110101011101100100000100111110000111100011010011101\
01000100001101101000010000110001010100100000011000101101111011001111\
00001111000000100010010100101010101000100111111101011110111110110100\
10111001010010000111010100101000111111000010101001110100101110101100\
00001010011001101001101110100110011001000010101101011011010001000001\
10011101110001010001010101000001001100011011111010101110010101010100\
10101101110111101101010111101011010110101101000100110111011001111100\
10111110001101110011101110000101111010010110010110100100110001010011\
11011110000011110001001001111100111001011111010101100000000100111111\
00001000110011010100110000000000000110011011110001110101110100001010\
01000000101100101001001100110010111010000001001010011111011000101111\
10010010011101010111011110110110101111110100110011011011100101100001\
00111101111010101000010001000010011000100001100100011011110000100110\
00010100010100001011101010111100111001110000001101011001100001011110\
00110111011111000010101001110110111010100101001110011100101000110011\
11010010100100011000111001011111001000111010000111100000111001111011\
01011010101011110010101000011111001100111111111011010110000100111001\
01001011100111101101100110000111010001101101011001010101000110110011\
01011100110000100001001101011010100111111110010110110101000010001010\
01010110100100110111101111110010011011101011010000000110100100001010\
00001010000100010101010010110001011000100100010010011011111010110101\
01100010011101111110000100010100010000010100101100111100001010110001\
01111001001011100011001111001111000100011110100101010010100000101000\
11010100000101110000000101001001100111001110000010001101011100000011\
00011001011100011111111010101111101010000000010001001010010111010000\
00100011100101111111000100111111001110110001000100111001010010001011\
11000111010010110011001001000011111110001001110100001001110011110001\
00001101000010000001011100101101000111011110101101100010100110110110\
00010001100010010101010000011110111000001000110111001110000100011100\
00101110111001111101101010111101010100001100111000000011110011010011\
00011011101010010011010001111010100101000101011110110101001111111101\
11001000111100010100100101001100111111111010011110111000111000000110\
00110000110100111000110101101111101000000111011100000010101010111000\
11010000001110111010000110101111000111011101011011011011000100010101\
00101000011110100011011001001010100101100110100100110100001110001100\
00110001000000001111011110101111001010101001100011101110110101000001\
01110000000101101001011010110100011010100001100100000111000110101111\
01101101001001011111010100001001101100010101100100011001000011111001\
01000011001001110000001001111110110000001110011111000001001001001100\
01001110100111000110011110010011011101000010101110010110001101000000\
11110001000111000111011101010000011011011110111010100110100101001011\
11100000011011011101100010011010010000001011101011100100001100011110\
11111110011101010101110000100110001100001011110010110101101101010111\
01000111100000010010010011011110100100111111011001001001001010001011\
01110011001001000101011100100010011001000010001110111011110010110111\
01010101000000110010110000101100001111101001001101100110001010001101\
11111110000001000101001101100010001001010010010111100000101000011110\
10100101011100100010001011010011110001001011101000000101111010000000\
11000000011000101101010111010111101000101011100100000010101000111011\
```

```
100111010010111000101001011000010101100011010110001011101000000000\
001110000100010011111001010110111001001001000110000101010010010110011\
111000101100111010100010100111111100101100011000111100111100000001100\
111010000111001010111001110100110000000000111110011011011111010111111\
001111000000100010010001110010110101111100110100001100100000100000101\
001110001111001101110101011110111011110010101101011001111101101000011\
000001000000101001010011100111011110101111101011000000111110000011111\
100100010100010100111110100100111111010110000010001110011010001011110\
111001110000000011000101111011010000000011100010001011000100101010010\
110110111000000000001001011110100011001101000111011001111111110100101\
111100000110011011000011000100111010101001010010000001111100010010010\
110110110111111000011110101001101010110001101111000001010100100100001\
000101111101101111110001111101111011101111001010100011100000110011110\
010111110100010011111110100100011110001110011010110011001000100100011\
010000100000000001100000011010010010110010101001001010010011111101001\
010010010011000101101111101100001011011111000000011110101101101101101\
101001110011101111010010110100111000000011011111101001111110111000011\
111000000011010011100010111111011011101000001000100100111100000110110\
011111101101001101101110011011010111010110111001010101011101001001010\
010100011011110111111110110110000101010110101001110010110010001100011\
100000000001100110111100100110000011001011111101000101110000010001101\
111011110110001101011100011101001100010000100110011011100100000010011\
001001011110000010000110110000110101001111011100000101100111101011000\
111101110111100001100000000001101101101101111110101010011100100111000\
011111100101111110101001011011101101110101001001001111101001001000001\
011010000110111111100100110010100001100101010010010110000010011000001\
000010011001010111010110110111010001001100110100100100010001101001000\
001001100101011001001111100010101111001111010011000101110010101101111\
101011111001000001000000001011100110101101100000101100110011000100101\
010011010011011001100110000110011001101010010110001000110110111110011\
111111010011000000011100000110010101011111010000000011010101101101000\
111100011000110110100110110101101110111010000100010011011001100011000\
011110010011101101110010001110010001101101011000011011010000000100011\
100010010000000101110000100111001101100001000010010000101101111110101\
101011010100110100110100001111110111110100101100000011011111011110001\
010111010010100100001110111111001111001010000111100000111001111101001\
111000010110000010110111011101111110110010010100010110000010010000001\
100010100100000101000010111111000
```

Supongamos que este ransomware hace uso del RSA de longitud 1024 bits. Entonces, debe dividir la anterior cadena en bloques de longitud menor o igual que 1024 bits. Para asegurarse de que la longitud de estos bloques es un número coprimo con cada uno de los primos de la base, los toma de longitud 1023 bits. Así, obtiene los siguientes ocho bloques de esa longitud y un bloque restante de longitud 8 bits.

```
Bloque1 = \
110001110000101100100110011000010100000010111110000000001000010110000\
100001111101001001011101000110001101100110000101110111100011100111110\
000010010111000111101101001001110011111100100110000001101101100011000\
110100111100001000100010110101111000111010110001011001000101111101011\
```

```
100101011110111100000110010001011101001110011000101011000011010111100\
110111111000100110111001010100101000011101010010100110111110111111101\
011000001111000100010100101011101101010111110000000000011000110011000\
111100000100101010111000010111110101000010111011000000011011011001100\
010000011000100000111011001110011111010101000000111001100000110110100\
111000101001100101011110100111001110101100001111000000100111010110010\
101110011110000111000100000001011000001010001101110101001010010010010\
010111110000011100101111111001011111000111010000010011011100110011000\
101100011110011110001101101100001111010000010010101010100001111011100\
101000010001010011110001000010100010100000000000010000010010110110000\
000001110100100110100001111011111001101111001100110001010010011010000\
100
Bloque2 = \
101011010011000001100110101000010001101111011011010001011101000101100\
000111001001111011011100111101100100011010000100011100100110100111010\
111110000111000110000100011100001010001001100100010110100101011011010\
101101101100101011100100100100011011100001101101110101011000111110100\
101010110000110011001101010001111001101101001011000000111001011100000\
001100101111100111110100011010110010010010100111001101010100101101000\
101101101011010011001001010100110010010111110011010011001010110100000\
101111111001010110101011000110110100111101010110011001011001001000011\
110011010000100101110101100011001100110000001111101110001011010100100\
101100110010011100011011100101011011000110101100000101001111000001001\
111001100010100001000110010100010011111111101011100000100101011011010\
000101011100010101010001000111101001110110000100110011000110100010110\
000000000110110001000000101010101110100000100011000101011001000110110\
010110101101000101101100000111101001111001000111001101101010010000100\
100111111111000111111110110111110000110000011101110000010001110001101\
100
Bloque3 = \
111010100000110000100011111100101100011100111001001100010100110110100\
001011101101111110001101010111010010000100010011111100001000000010100\
110000110011000001001011101101100000101110100101100011111110100101100\
011100100011110010100001001010110011001000100111011100001110001011100\
101100010101101010111011001000001001111100001111000110100111010100010\
000011011010000100001100010101001000000110001011011110110011110000110\
110000001000100101001010101010001001111111010111101111101101001011100\
010100100001110101001010001111110000101010011101001011101011000000010\
100110011010011011101001100110010000101011010110110100010000011001110\
011100010100010101010000100110001101111101010111001010101010001010110\
011101111011010101111010110101101101100010011011101100111110010111100\
100011011100111011100001011110100101100101101001001100010100111101110\
100000111100010010011111001110010111110101011000000001001111110000010\
001100110101001100000000000011001101111000111010111010000101000100000\
001011001010010011001100101110100000100101001111011000101111010010010\
100
Bloque4 = \
```

```
111010101110111101101101011111101001100110110111001011000010011110111\
110101010000100010000100110001000011001000110111100001001100001010001\
101000010111010101111001110011100000011010110011000010111100011011101\
111110000101010011101101110101001010011100111001010001100111101001011\
001000110001110010111110010001110100001111000001110011111011010110101\
010111100101010000111110011001111111101101011000010011110010100101111\
001111011011001100001110100011011010110010101010001101100110101110011\
100001000010011010110101001111111100101101101010000100010100101011011\
001001101111011111100100110111010110100000001101001000010100000101001\
001000101010100101100010110001001000100100110111111010110101011000100\
111011111100001000101000100000101001011001111000010101100010111100101\
010111000110011110011110001000111101001010100101000001010001101010001\
001011100000001010010011001110011100000100011010111000000110001100101\
111000111111110101011111010100000000010001001010010111010000010000111\
001011111110001001111110011101100010001001110010100100010111100010111\
100
\
Bloque5 = \
101100110010010000111111100010011101000010011100111110001000011010000\
100000010111001011010001110111010110110001010011011011010001000110000\
100101010100000111101110000010001101110011100001000111000010111011110\
011111011010101111010101000011001110000000111100110100110001101110101\
100100110100011110101001010001010111101101010011111111011100100011111\
000101001001010011001111111111101001111011100011100000110001100001101\
001110001101011011111010000001110111000000101010101110001101000000011\
101110100001101011110001110111010110110110110001000101010010100001111\
101000110110010010101001011001101001001101000011100011000011000100000\
000011110111101011110010101010011000111011101101010000010111000000001\
011010010110101101000110101000001100100000111000110101111011011010010\
010111110101000010011011000101011001000110010000111110010100001100010\
011100000010011111101100000011100111110000010010010011000100111010001\
110001100111100100110111010000101011100101100011010000001111000100001\
110001110111010100000110110111101110100110100101001011111000000110\
110
\
Bloque6 = \
111011000100110100100000010111010111001000011000111101111111001110101\
101011100000100110001100001011110010110101101101010111010001111000000\
100100100110111110100100111111011001001001001010001011011100110010010\
001010111001000100110010000100011101110111100101101110101010000001\
100101100001011000011111010010011011001100010100011011111111000000010\
001010011011000100010010100100101111000001010000111101010010101110001\
000100010111010011110001001011101000000101111010000000110000000110001\
011010101110101111010001010111001000000101010001110111001110100101111\
000101001011000010101100011010110001011101000000000000011100001000010\
011111001010110111001001001000110000101010100101100111110001011001111\
010100001010011111110010110001100011110011110000001100111010000111001\
010111001110100110000000000111110011011011111010111110011110000000100\
010010001110010110101111100110100001100100001000001010011100011111001\
101110101011110111011100101011010110011111011010000110000010000000101\
```

```
00101001110011101111010111110101100000011111000001111100100010100010\
100

Bloque7 = \
11111010010011111110101100000100011100110100010111011100111000000011\
00010111101101000000001110001000101100010010101010110110111000000000\
00100101111010001100110100011101100111111110100101111100000110011011\
00001100010011101010100101001000000111110001001010110110110111111000\
01111010100110101011000110111100000101010010010001000101111101101111\
11000111110111101110111100101010001110000011001110010111110100010011\
11111010010001111000111001101011001100100010010011010000100000000001\
10000001101001001011001010100100101001001111101001010010010011000101\
10111110110000101101111100000001111010110110110101101001110011101111\
01001011010011100000001101111110100111111011100011111000000011010011\
10001011111011011101000001000100100111100000110110011111101101001101\
10111001101101011101011011100101010101110100100101010100011011110111\
11110110110000101010110101001110010110010001100011100000000001100110\
11110010011000001100101111110100010111000001000101111011110110001101\
01110001110100110001000010011001101110010000001011001001011110000010\
000

Bloque8 = \
11011000011010100111101110000010110011110101100111101110111100001100\
00000000110110110110111111010101001110010011100011111100101111110101\
00101101110110111010100100100111110100100100001011010000110111111100\
10011001010000110010101001001011000001001100001000010011001010111010\
11011011101000100110011010010010001000110100100001001100101011001001\
11110001010111100111101001100010111001010110111101011111001000001000\
00000101110011010110110000010110011001100010010101001101001100110100\
11000011001100110101001011000100011011011111011111111010011000000011\
10000011001010101111101000000001101010110110100111100011000110110100\
11011010110110111010000100010011011001100011000011110010011101101110\
01000111001000110110101100001101101000000100011100010010000001011100\
00100111001101100001000010010000101101111110101101011010100110100110\
10000111111011111010010110000011011111011110001010111010010100100001\
11011111001111001010000111100000111001111101001111000010110000010110\
11101110111110110010010100010110000010010000001001010010000010100001\
011
```

A continuación, para cifrar cada uno de esos bloques haciendo uso del RSA, convierte las cadenas en números decimales, que son los que va a cifrar.

```
a1 = \
69886568603280157880073385528299599339945789224219386686987379322222\
70416974777256792045992549970173898643139987577850248839330687415530\
89227828133241843998794410557537507396579339626540164523591690996868\
35060963061093729031438792136081008334185629257220718397257910182452\
82615819504142005581658753205918598 8
```

```
7
8  a2 = \
9  6080874901956847135316133325483759324791730900004474861566459458127 8\
10 9537172895680568309470797954917458401198534875107259211326304415490 1\
11 5674965919598022735356289674690859956229714273712004963118204799598 2\
12 1292654784132483052838500176904839834460180670974119168025691890449 1\
13 3220365121784595962113577686024264 76
14
15 a3 = \
16 8217684502130575929827154909093389402724514082242581038401733526372 7\
17 0811030875842006442698643133416405310269705390297322654418858536803 3\
18 4003354779093956898978776071286765316451762772664247614637644901107 5\
19 0217750778738230134109166117348848682984003407277618814942598042506 1\
20 9207122645364350873740424518089270 12
21
22 a4 = \
23 8248857658903751913493573140143760309819135761668177569956018757293 2\
24 7921673881388454286062711619941817634292410110733361688126477305178 7\
25 3794622508200791886365283755472198711833554388874548357754668540948 2\
26 9948469082933447247610355984774269436306308863216117876720873790477 2\
27 3310944370184199299851353625676176 52
28
29 a5 = \
30 6289875285810758237977266949223272686534165403294199714507016295935 1\
31 3103795012562429525835645762673621924916385984279264073577460308006 7\
32 4080040523977794856528373919699114059807189354593495698194175737573 5\
33 6237036726904887292893314161728570655324133541306710084191404872283 2\
34 1419166041899759773133216706518016 54
35
36 a6 = \
37 8296819922038366310065090992532971297321013487651254366306102351320 8\
38 2183069228046083494883069956834701277807536198283583346221892113614 5\
39 7878554803197503029971896503011639386904445112211340575316670111364 7\
40 7001998810317085069185793668925996638671593445209547397384711519897 6\
41 7429952549300283569588435633673515 72
42
43 a7 = \
44 8788748275153763504455346490043623185148078814744549217954780761041 0\
45 7546414489718644681771297487683986145921321808540752332379286190745 9\
46 7988316689818021439917936145140338312384756130861854900505527590108 0\
47 3386861447176576621609902845416687359384476011638800774967245410119 8\
48 0236638267113180906784811061433292 96
49
50 a8 = \
51 7598622313019861756532582901473060973571198288756842843770480258933 9\
52 1627112129422056017924934594004700513142021930797026521811171863909 9\
53 6596145359979833456945415283289175551790341149076910496309388555627 4\
54 4336154238057188504211562596940973189103160849544864719206708032627 9\
55 3233535819127210579150167544289702 51
56
```

```
a9 = 226
```

Ahora sí, cifra estos números con el RSA del capítulo anterior y con la clave pública (e, n), donde

```
e = 65537

n = \
8901686781795509927565852625863866312803592135915260212942120173486 2\
5927784226275641413184313433073611981388019475349186541540662470424 8\
1275478728775831229019673980365473664533367340351892041078384373913 1\
4075797039640150150930190066145613835033429627736821333853836709957 4\
4209640917712076118167013946837364059798343047514594097884993473226 3\
2907035717839702487762470592719494404620622198308390415845418776834 6\
8686361044545772272629250421492804934543782813006608392467515604142 4\
7063659031825291247576162224062859092938425723788042861591745098255 0\
7757758260655706989209970131531198644618020428513393019508638323562 9\
30061
```

La clave privada es (d, n), con

```
d = \
4604395688177456872218057557778952410664646990484633198933807341822 3\
4010192066870896871470086227135242268597471232965532059320489450004 2\
8349748286109707536697406476653023357096245166403539806529382667672 9\
4136982831786036131748211133440196581988773226584227328017932017499 2\
3447252963509493390447283302315330333404877084023009286179033054672 4\
7777878828844566209764107937310787911365052182739072087561807138014 5\
0833824717132085330274620956321303504194659045771376759664068378356 6\
4120579980751073788658185366551093215531407253764348099984528026312 9\
3510835162267215932572875317442647401011722730533967164777625256217 6\
94177
```

Esta clave privada la mantiene en su poder el atacante y no se la proporciona a la víctima hasta que esta pague el rescate.

Una vez cifrados, se obtiene lo siguiente:

```
c1 = \
6200281583473679898778729140805107561822637178079379202989963704760 1\
1329750842114394066758300606151272970754942207275112151467966225888 4\
1333822882106068879568855742787897363153866420481743036262614763041 7\
5363338339616496011408089443734133517138105978853124514352869223216 1\
9042484807892239628903527504753966428536290464685427605227081196361 1\
2368897004934308635960819603002603677541516116844833309448909455075 4\
7488312367648814546851892515383440875049409949102520927721798200800 6\
2183816394363745608833872038460967396583127623102100074698590846067 0\
2961202799318611588459068508518836826059985797367924070872802886923 8\
96500

c2 = \
```

```
296644867408614046118296152506805186484104789219246690806137153042824\
492025647308861422243590300170454866187433647894780144731642725515584\
364045572805850919098137942073434674196141393025028937158345825252494\
546757251444221674762557267058167449062820185389219654102819172225994\
884793154850725518856838495940453040445107390096540500178007890975524\
112774548357569326681963345808592898258703732152096689141627936015194\
112260201464884967748095529924270520972613677192547795685498580775004\
417119563018879573321447814146454636626902162761797584749641937783644\
440328171617043990705043895505214801771979642560416528816380694351334\
76304

c3 = \
123887652797953006172937788823149021492314973378551993637680445114784\
641519392246389826521533457819738400830763255077799001322028020325774\
623891100532722739485682285695934662572774357183205791035967432405664\
748692846328909849467328153636015972600462986879592180706844205453494\
046962673953661584603049913963200813516848230037414816044020831886084\
950102810536012473682018481468839342629103268529382037352494909548864\
091573896164818784941340533723471983792472544563552771346904769259684\
439315969840083606022343650409208939828095611741606534229218994424924\
095529896228593163990891570248977162734496919280914830352790289812294\
79829

c4 = \
218763408700572447587980282931160932170728295850483645811115402030184\
519889404214853822131740737700001102209229662426312667933537277605984\
340400958534547410550388803454260542074129843223473133009167228339144\
296896358374652594578086654085813157336379738141436720406241278419804\
028367410402128382487264394787697183893378480763292187805261714593104\
940511260485945738760671935225863518339604223821727656156282341117954\
556735771880970373391636417911740755873000430296193131741355371559824\
661052101682367798187660014044931358122797904875665001373370320722984\
420106745322122057161893473393938037329996721687745663979994150577044\
36162

c5 = \
452204242436320824603465903428418684049448903424461313126643635803154\
199302779256128037410858547431148144982431986366013297008779936918994\
737182298206595247154380211268238123067751402175359745416502830074484\
230651758380297708803459389686342296040212734487747475740356720970624\
805356693179235982793271186817410048658512469411964818733106046610114\
268058769262002305174663238967475736199300530104211115206657264811624\
599519196665682251664717281203758236660609052694225904308187763384158\
565915985454634684961613010738921203397454937510854617756822020856074\
294430910947472912374127249349631381029286420293143996515788793457374\
46605

c6 = \
803204954244503904396389264562286499071975858936552021140137802052344\
509213418376963776481082696466454492387845226059706863656191664185474\
```

```
031350529521389864000291770344886493971827479021820045100176927054325\
550583317507624316055462759583068315313745593024618182915765569886375\
361082957873093466686238782381953285148716029436513925819986222623645\
353335607182720367318164484136684529772878414705968270601542446124715\
033306247798968927416758465717037897648920231989759901117491144166815\
032908639374872551148050718817635316230779357017279127513582465851555\
123996044523337972740046016639166131490586316191097908929000751907785\
49991

c7 = \
286150188437774481019035523092329786294824762738007229690431675830355\
156276419997107200634837995573067871500688110146902028107365816745655\
670693708217530377363989923093485482817745227906958403537284434264275\
847951388794263543293529594150906468238346156899124761227086597163355\
118996571195518009366905392430953587756238795685970413320252250788065\
333496830853031115802900010423952393172811628414219680554311358437165\
069020138429327772572807250115377686329041196660434214318027997995925\
515479626901152759429456255962338761972958922689744399766741682579845\
182934492886911565142951193196527519553223956101513022367603292519914\
4845

c8 = \
330779366643798709521203939701619974735075892299768179960807887116975\
592430775649640775828549151681695353491616942205664416002358974711205\
968285758298116777668412116916991713597578925421460944513863365579385\
130189841413534453040664116572594821310368342974305291986660696596335\
672704326692368136616690903686692656195631844389047329237118551184105\
930044949446122331715268581247727705856406170101135192388643860839935\
845113394687741560197022516641211204841452145710804857882640205667135\
743026115824327994717841752221564796694892021830716072692818085348935\
746362841277653615783076804009975514806186419086447321405222081687125\
76232

c9 = \
669787486236063060317088279222483217255283331049982662737834032223675\
726540863289746133781714489229201761381881196583149693665134426321865\
111835963235841851504241351058089613642029220483216794906494275332395\
333365941845641679868366285933244598423601331672668905676546963778425\
421537849937790182331596089956653146363260725572677422646363879343805\
302332883259082901107707540307691459140844610186055088734571362753715\
687949303117024386342822404000048757515682387403948840705178412801815\
944708387827804236685837949052523772557140657704949690717068936848515\
695578016950154781056664624692506165992669641452951133036164045347895\
88348
```

Estos números son el resultado del cifrado de cada uno de los bloques anteriores.

El siguiente paso es transformar en binario cada uno de estos números y unir los resultados formando una única cadena. Esta cadena sería la que sustituiría a la del archivo original. Descifrar esta cadena equivaldría a combatir con éxito el RSA, algo que, como ya se ha mencionado anteriormente, no es viable actualmente. Por tanto, la única forma que tendría la víctima de

recuperar la información sería pagar el rescate.

4.2 Criptotroyano frente a un análisis estático

Recordemos que un troyano es un malware que accede al ordenador tras engañar al usuario haciéndose pasar por otro tipo de archivo o programa y que, entre otras funciones, puede robar información y almacenarla en su interior para, después, enviarla al atacante. Además, como ya vimos en el capítulo anterior, un malware debe estar preparado para defenderse de los analistas. Si un analista nota la presencia del troyano, pone en marcha un análisis estático para descubrir qué información ha sido robada. Ante esto, el troyano activa su módulo de defensa y cifra esa información con un criptosistema de clave pública, lo que lo convierte en un criptotroyano.

Un criptotroyano puede tener como objetivo robar un determinado tipo de información, como puede ser la de los mensajes de algún correo electrónico o la relacionada con transacciones económicas online. Consideremos un ejemplo de criptotroyano que haga uso de ElGamal y que se centre en la banca electrónica y supongamos que roba la clave de acceso a ella de alguien. Veamos cómo actuaría:

Sea $m = 692524$ la clave de acceso obtenida, que, en binario, quedaría como sigue:

```
Clave = 10101001000100101100
```

Comienza aplicando el criptosistema de ElGamal con clave pública (g, p, k), donde

```
g = 13

p = \
428690182440548306053861161418597613360473088436836740397527645482 01\
105881972791722480444850600066724337909990028204001720686759230282 23\
449403957161184820237161931945805515911627925288780250419530633407 58\
593658087444111356840631322693845996551081144145495224640295706381 87\
299703862864618753433938463421074832 3

k = \
314881713039402238428761808469304151435040212663072417164398906738 58\
008102611414325855183246411332961085223919395763677706852167950506 57\
945520162984736210871584847352348035957316205566933229956310566610 26\
362512770820662710110512565677173315656421248497830217732606922029 13\
494578030734338688952258789598515649 7
```

Así, el cifrado de m queda compuesto por (h, d), donde

```
h = \
245760185865025427587196868799433645098964243136687742251685397750 22\
319767853364696784052316103192482922588299381262090774146224379963 15\
056544448501072409645744518358545717633351495061382210746814331087 85\
523242449469547082594357120232741380413263494811213842966586202050 97\
927126058765504577616823662601582503 7

```

```
d = \
266108634752590087472780570607179900126573662958050266228264931678 83\
130555559890206652212763799181848828115501120410284975404241160179 65\
672709951164281179072657464664008783088107573663581280953526017216 04\
228481734038863465705232099225873008396798886363295759552812006913 73\
143010978853126179395789980488668620 4
```

Entonces, el criptotroyano almacena esto en su interior en código binario y, posteriormente, lo envía al atacante.

```
h' = \
101011101111110010110000000001000110100010010110001000100110101001 01\
011000110110110100001000100110100001111110011011111100001100111001 01\
010101100101001101100111100100001001110001110000010111100001001011 00\
001011100001100110101111000000011100000001101111000011000011010101 01\
000010011000000011110010011100110011101111100000010101010111010111 11\
100100011100000010010000101001010110111011101110111000010011110100 00\
101001111101010000100001000001101110001010011000110011000101101111 10\
100001111111000010001111010001110010001110011111101100000000100010 01\
001001001000100011100101101001010101101111111111101101110101010001 10\
101101111000111001001111000101111000100000111011111101000000001010 00\
110000010001111101110111110101111010001000111111000101001001001010 11\
110100110011111101101111111011100000111011110110101111000010110111 11\
111110000101000011010011111101101011110100010000110100100110000110 00\
110111000100001010011001110110101000111111101111011000001000011100 00\
101011100100100010101101011000111000000110101110000100010001011001 00\
01101

d' = \
101111010111100111000011010111101000010001100101000010011000000110 10\
101001100011001101011110001101000100101111111110010001111100100110 01\
101000111010000111011011011010101100010110110100111111010100011001 01\
010001010100011000100100010010010000100010001101110011000011001101 01\
101100111101010011000100110100001110110101100100111000100101001010 01\
001100010111001111111011111001011001101010111111001011111111011111 00\
000010100010000100101001001100010011101010100110110010110011110010 10\
000000010001101010110110101111001011100001100010001010111010100010 11\
101000000011111001010100111000101001101101000000011001110000110010 10\
111010011100000010000111100110000010100100010100010111010101100110 11\
101111010111111101110000010001100100000101111001001010001000110100 01\
111000000111100000000000100100111111011101001000010100100111011110 10\
111100001101101011101110100011101000010001011100110011111010001000 10\
110000011001010111111001010001111101010000110111100011010010001000 10\
000111001010110000101010011001111001000011111000110010011110110011 11\
100
```

Claramente, este código es muy distinto al inicial. Sin embargo, el atacante, quien es el único poseedor de la clave privada a, puede descifrar m sin problema alguno, tal y como podemos ver a continuación:

```
a = \
20643080413172394834383410903233381099520497706556917968086823739322\
59007115733920835183709859690441924208080203375425512072992756725995\
19387611451074046528054009573333657990637684513401211447669314055379\
20477858580518915637097211705281007058188856455424146366932280497231\
61807318494473349507149661701179071 60

Mod[PowerMod[h, -a, p]*d, p] = 692524
```

Si un analista se percatara de ello, no podría hacer nada al respecto, pues eso supondría romper un criptosistema de clave pública sin conocer la clave privada, algo imposible.

4.3 Criptotroyano frente a un análisis dinámico

Ya hemos visto cómo actúa un criptotroyano frente a un análisis estático, pero si el analista descubre el troyano y, en lugar de ver qué información ha robado ya, observa qué está haciendo, puede poner en marcha un análisis dinámico. Entonces, un criptotroyano que haga uso del criptosistema de ElGamal puede aplicar su propiedad de reencriptado para evitar que el analista descubra su funcionamiento. Veamos en qué consiste esta propiedad:

1. Se toma un número aleatorio $b_1 \in \mathbb{Z}$ tal que $1 < b_1 < n-1$.
2. Una vez obtenidos h y d con ElGamal, se calcula:
   - $h_1 = g^{b_1} h$
   - $d_1 = k^{b_1} d$

Esto consigue cambiar, aparentemente, la información almacenada en el criptotroyano, creando confusión en el analista en cuanto a qué información está siendo robada, pues da la impresión de estar cifrando nueva información. Sin embargo, en realidad es la misma. Además, la misma clave privada a permite descifrarla, tal y como podemos ver a continuación:

$$h_1^{-a} d_1 = \left(g^{b_1} h\right)^{-a} k^{b_1} d = \left(g^{-ab_1} h^{-a}\right)(g^a)^{b_1} d = \left(g^{-ab_1} g^{ab_1}\right)(h^{-a} d) = h^{-a} d = m$$

El criptotroyano también puede estar programado para realizar sucesivamente este reencriptado, por ejemplo cada cierta cantidad de tiempo determinada de forma aleatoria, logrando, así, confundir aún más al analista.

Veamos un ejemplo:

Supongamos que se quiere cifrar la clave de banca electrónica $m = 457903$, que, en código binario, quedaría así:

```
Clave = 1101111110010101111
```

Tras aplicar ElGamal obtendríamos las claves pública (g, p, k) y privada a y el cifrado (h, d), donde:

```
g = 13

p = \
619798835774806962413355823509503016529826386612810271197709155136 99\
100924067970207074349912789072561910152914141177401665778098163414 08\
395605848047715129144985700166575285257263548228812680760925040876 43\
983795987900545113870210162240480901159233179369569745246527268815 62\
251306001713835385062118832784489598 7

k = \
179855063383227761245964045125512446253494944255285440442568398058 94\
447423303748368898831052341470518658404770966026350536303671157497 11\
626708627666804419885595533904960770730984955265994931502450254741 11\
085793282598247378759775990728413974297992763946997862844613855434 06\
175125711325984276839317552416933238 8

a = \
117088272005171592187517159724159003012541404968105875289036315877 73\
428347160568147239812603026283771541236522507919966364341750376824 29\
581813415702227649448164015263504714307644570742712617979856769395 84\
155126337643325811297188886726451211921117284791158027452239891876 08\
477233665041816541922316753930692817

h = \
328636404091493005924122460126101696416251613493614540406982557816 54\
490570989038831632821926424944612201427365958047441172893583518615 89\
035719794587596862710810395559238796769479748520744847223486201246 93\
148194100449214027891457526123792339523113776859234705956069151974 26\
782531227390028281538590916148117 0012

d = \
880649027855929440606748756018170279025108840027963037192632610546 24\
502407715310764714586529289664306494781579572598948952572817482359 15\
145650574000515406459991663024257262135748722313506008498016653799 78\
735675228377757970069195344103738445583618739448241563641237560640 40\
453539915872360792105888395879712523
```

El cifrado en código binario sería el siguiente:

```
h' = \
111010011111111100110011111111011100100110100000010010000111010011011\
100011011110000010100010000010011100010110000010111100111011100000000\
110000111001111110111111000000010111111011110000010011001001011001 00\
110111011100101100101110101111011010000001000010010110111001001100 11\
100001101101010000101000110111010000000110001010001010001101111100 10\
110011000001101111100111111010111011000010110011101001010010110101 11\
000000001010000101110111101101001000111111000010010110101100001111 11\
011101011011111111011101101100111010001000011111011110000111100110 10\
000101010010010101001110111000011101001011111000010011010010111010 11\
```

```
00000000110000000100110000000001000011110000001010110011000110111101\
10101000110101101100011000000111100110100000101001010110110010100011\
01101111100101101100001100101000111100100001011101100110011001000001\
00000100101001001111100000011001100100100011101000111100111111011010\
00111000010011110100011000111001011110010001100111010111011001010000\
01101100010001010010110000100001000000101000001010100001101101100010\
11100

d' = \
11111010110100010011000111110111101001010001111110011011101010001011\
01101100111000101000011001111000111011111011000011111010011011001111\
01001111111101111100001011101010100101001000101000101100011111001111\
00111101100010110111010010110101011011110100111111010100000011101101\
00111101001000101100101101111010011100010010100111011101011010011101\
10110011111101011111110011010111101110011001011111000001010101100111\
11011111010110110111000011100101101111110001101000001000000010100100\
00111001111101000001110100101011011011100110000111110111101010010110\
00101010101111010011101100100110011010110010100110010000011001000001\
10110110111110001111101010001010001111110000010001001011100011011001\
01100011011010101010001000000111100101000101011001100010000001001000\
01111100001111011010100111010001110001100011110000011110011110000110\
01100001101011011111010111110011010010000001001110011010111101000110\
11111101000100100010001000011100111001101111110110000111111001000011\
00011010101011101001111111011101100000101010110010011111000001110000 1\
011
```

Ahora se realiza el reencriptado, obteniéndose lo siguiente:

```
b1 = \
40982267176646004009047030541921769585531257788889503182463898447468\
85121188886015764623254433970308556560570890853119830845347028122615\
04808634358302220394769598948504928451852104733926827436508235834594\
20118192673940931743513337807945854570301900935073202062202864296186\
58730842683228270228130275548722715 79

h1 = Mod[h*PowerMod[g, b1, p], p] = \
49451909580571815076964176721875992866980297069049039408366949294317\
59670260626013059993349883381088913546796142169469037948600030763362\
48867762022380936870901820326351131624025071183199637164504114294223\
32006530844247149959196670487753284325895238072195833407254570006929\
86496262640612968281602103299385845 38

d1 = Mod[d*PowerMod[k, b1, p], p] = \
41417935992750668855619111026504096977227546555898626512614619429003\
86416151944608175806214724137165655388584494788917242901236193153964\
80234353431094531165529182977578063316611035231264609943865409529816\
18743421770216748829885919935925770088562917249480197872006800647644\
48789782498441757785699494940857882 12
```

En binario se tendría lo siguiente:

```
h1' = \
10110000000011011111110100000100111011101000011010001111101111101101\
11101011111011001100111110001110101110100110010010101100000100100110\
01111100100100111100111111001110011110101001011001110110000001101011\
01110100011110110101101100011001010010010110001111111101110100100011\
11000101101110101111010100111100010111010011110111010100110011111100\
10001100001100011100001111101011110010110101010011100001010111100111\
11100101110010111001010011101011010011101000110001110110110001010101\
11011000110010000111100111111011000000010010100110001011110110010001\
01001111100110101111110011100010101111010111011011100100111001101010\
00101011101011110111101001101011101110100100000110010111111100010111\
00011110011011000100010110000001110010101000111011100100101001001111\
10000110011011110100100011100100111111010110010010100110010000101111\
00101010101001001001011110110000111000000001110110000100011001010001\
01010011100111001000100111111111011000101010111001011111001010111000\
00100001010101011101111001100111000000001100010101011000110011000111\
011010
```

```
d1' = \
10010011011100111110011100110100001100011010000101100000111010101100\
00000010100001101111110000010100100110110011100001011001010110100000\
00101110011001110111011110011011010000110001100010100100110110001100\
10001100011111000001011000000011100111110101100010010110110100111011\
10000001111101111101010000000100011110100010011110101001101010111111\
11001011110011100111010010010011101111001000110110101110010101001000\
01001110101010101110110010110011110011111110010010100000110110101010\
00110100001010101101010101110110000001001000011011100111011111110001\
11110011111010100101100010010010000011110000010110101101100010111000\
00111101000001101010110000111001011001010101000101110111100101011101\
01001111001110000110111110101010001100010001001011001001100100100011\
10001001101010101110001110100001000010100010011111011001110010100010\
01100111101101011001010111100011011100100110000011010101100100000001\
10101110010101001101001100100100111001101001110010110010111010111011\
10111011110100110001100111111100111110101001101000001011011010001000\
110100
```

Como podemos observar, el contenido va cambiando, confundiendo al analista. Sin embargo, haciendo uso de la clave privada, el atacante puede descifrarlo sin problema:

```
Mod[PowerMod[h1, -a, p]*d1, p] = 457903
```

4.4 Criptocontador

Como ya sabemos, el objetivo de un virus es propagarse infectando otros archivos. Pero esto no significa que estén activos y atacando. De hecho, se suele introducir en el código del virus un contador para controlar el número de archivos infectados y, una vez alcanzada una determinada cantidad, iniciar el ataque.

Esto es algo que los analistas intentan descubrir. Por tanto, el atacante dota al virus de un módulo de defensa que oculta el contador, transformándolo en lo que se conoce como **criptocontador**.

A continuación, se presenta un ejemplo de criptocontador que hace uso del criptosistema de ElGamal:

Supongamos que se tienen la clave privada a y la clave pública $(g, n, k) = (g, n, g^a)$. Para cifrar la primera etapa de infección, se toma un $b_1 \in \mathbb{Z}$ aleatorio y se calcula

$$(h_1, d_1) = \left(g^{b_1}, k^{b_1} g\right)$$

Haciendo uso de a, se descifra sin problema:

$$h_1^{-a} d_1 = g^{-ab_1} k^{b_1} g = g^{-ab_1} g^{ab_1} g = g = g^1$$

Para cifrar la siguiente infección, se repite el mismo proceso. Se toma un $b_2 \in \mathbb{Z}$ aleatorio y se calcula

$$(h_2, d_2) = \left(h_1 g^{b_2}, d_1 k^{b_2} g\right)$$

De nuevo, se puede descifrar con a:

$$h_2^{-a} d_2 = h_1^{-a} g^{-ab_2} d_1 k^{b_2} g = (h_1^{-a} d_1)\left(g^{-ab_2} k^{b_2} g\right) = g g^{-ab_2} g^{ab_2} g = g^2$$

Veamos una infección más. Se toma un $b_3 \in \mathbb{Z}$ aleatorio y se calcula

$$(h_3, d_3) = \left(h_2 g^{b_3}, d_2 k^{b_3} g\right)$$

Se descifra mediante a:

$$h_3^{-a} d_3 = h_2^{-a} g^{-ab_3} d_2 k^{b_3} g = (h_2^{-a} d_2)\left(g^{-ab_3} k^{b_3} g\right) = g^2 g^{-ab_3} g^{ab_3} g = g^3$$

Nótese que si el virus alcanza una etapa de infección $i > 1$, se tiene que

$$(h_i, d_i) = \left(h_{i-1} g^{b_i}, d_{i-1} k^{b_i} g\right)$$

Nota. *La potencia de g indica la etapa de infección en la que se encuentra el virus.*

Como veremos a continuación en un ejemplo con Mathematica en el que tomaremos $G = \mathbb{Z}_p$, el descifrado proporciona un número en su forma entera, no en forma de potencia de g. Es por eso que el atacante suele calcular primero un gran número de sus potencias para, así, saber al instante en qué etapa de infección se encuentra el virus.

Supongamos que se tienen las claves pública (g, p, k) y privada a, donde

```
g = 5

p = \
3712040449833927518481075866841813211816952484508170691522914419708 1\
6385762889894790277241692195216127783067181789681468165337486778307 2\
1814034817293882871896897889404842957684410862883928425973493500272 3\
1988931319549413951262934532652083730967512768148116242787998353154 7\
0430488763064627140428641882817312787
```

```
k = \
13384459742941018227950481720893117641383431916480429462391738015320\
08126190020632781503554010624452020140974635671497728891146095003143\
45530704629881862620979660937763477514117712629771928434228272949819\
44552158878336957125614230849875622018124738165884859752906755720802\
59276196813618265040210334126442642 76

a = \
15021556415898151983134715404860147201923285944977055074458201471260\
38883595750672732497210271782533855695311616484362537524111945961088\
44360048331248135322889925534180485740063518077334948667343773323911\
56086435786322212217992807179578969267572327678400984295909153423888\
98921907760291031532523946367688514 29
```

El cifrado de la primera etapa de infección sería el siguiente:

```
{h1, d1} = {PowerMod[g, b1, p], Mod[g*PowerMod[k, b1, p], p]} = \
{13656109264353395272683667522705385573274803267449785312099592136 58\
56227408522046987974801765671433631889503175144478895304721717494304\
40943978317562264407954586585578634095550131256910606845277208771779\
22035234211795436040847942065904063321682156759854601896208568554865\
61611306577888712606675938662745385388, \
22427038917155052429657552306253486405988332965486379751549371780214\
17115580157440735817771658202453971506658299399756126807490423306795\
88729877899505810062831358846350355358611394561202364207566404249371\
01076313228322924831546363214968337963404780593565692871807577194629\
04566384628529741245806921836194830 19}
```

Como cabe esperar, tras descifrar se obtiene $5 = 5^1$, es decir, se encuentra en la primera etapa, como ya sabíamos.

```
Mod[PowerMod[h1, -a, p]*d1, p] = 5
```

Cifrando la siguiente etapa, se obtiene lo siguiente:

```
{h2, d2} = {Mod[h1*PowerMod[g, b2, p], p], Mod[d1*PowerMod[k, b2, p]*
   g, p]} = \
{17297856292687302415399989988018926870586888277890537401507023330 89\
59050030532816523918204434688005708957347127304484928372052426233407\
11368827215456022100205689171691500673686872143144409760800601034754\
88209503540941024015299049007065639117975237309601756797865171645723\
64019170128625352726618717158602971852, \
31828185545848910897445298575290806567636724012024564397126552853390\
81461434098066529325621860649961749864992013947043317616557581914602\
89583961613236733183617357838659841902796806566360135722416648819151\
59015181954285486692905819471766541441916542470926089668789379346356\
61852707585002611591017479449728652 0}
```

Tras descifrar, se tiene:

```
Mod[PowerMod[h2, -a, p]*d2, p] = 25
```

Veamos una etapa más:

```
{h3, d3} = {Mod[h2*PowerMod[g, b3, p], p], Mod[d2*PowerMod[k, b3, p]*
   g, p]} = \
{29384988842900694776686601816647460471240205991410458532282070709 98\
79213092398416080299922383256308344344095690105721707289038898499193\
16163913873027645306762608985494733659904117216256483803375009723826\
59811083495224606386225429644272340103087449764798157078043977739123\
01792884962092981098286321873009040 4, \
91910788070294313298520043329259918548629460399037487082665669309009\
37574686334875979460685443720863788179082937814624103388101965971263\
86679155228614511753082714242787072555278978608810280964274979883623\
64545663752872749899081297038918618191660798327205708132165683011254\
53362688431524183240414787708992719}
```

Descifrando, queda:

```
Mod[PowerMod[h3, -a, p]*d3, p] = 125
```

El atacante, al descifrar, como $125 = 5^3$, sabe que el virus se encuentra en la tercera etapa de infección.

4.5 Criptocontador de Paillier

A diferencia del criptocontador de ElGamal, el de Paillier proporciona directamente el valor del contador. Veamos cómo funciona:

Supongamos que se tienen las claves pública y privada n y $(n, \varphi(n))$, respectivamente. Si el contador presenta 1 archivo infectado, para cifrarlo se toma un $r_1 \in \mathbb{Z}^*$ aleatorio y se calcula

$$c_1 = (1+n)^1 r_1^n \ \left(mod\ n^2\right)$$

Cuando el contador aumenta en una unidad, se vuelve a cifrar tomando un $r_2 \in \mathbb{Z}^*$ aleatorio y calculando

$$c_2 = c1(1+n)r_2^n \ \left(mod\ n^2\right)$$

Continuando de esta forma, cuando el contador presenta i archivos infectados, se cifra tomando un $r_i \in \mathbb{Z}^*$ aleatorio y calculando

$$c_i = c_{i-1}(1+n)r_i^n \ \left(mod\ n^2\right)$$

Ahora realizaremos un ejemplo con Mathematica.

Nota. *Para que Mathematica pueda realizar los cálculos, tomaremos* $c_i = a_i \cdot b_i \ (mod\ n^2)$, *donde* $a_i = c_{i-1} \cdot (1+n) \ (mod\ n^2)$ *y* $b_i = r_i^n \ (mod\ n^2)$.

Supongamos que se tienen las claves pública n y privada $(n, \varphi(n))$, donde

```
n = \
5409208666935906335721984381240568833348817180189163039867420502026 0\
3507584668541417769025208694639931728933008322423509307190287576749 6\
1785889362348812903140773300475493807100598065676881503444228877731 1\
9059989476341777895419074498481700768782320951401455481898439487386 1\
8220239172328873908980179133068206063000748341719442228719514537461 5\
7774139105957406609014960827128305522852680935119617613152667559521 9\
2911881908785641193316156270329532206511127894303495115845556980437 8\
5231102224010277274161307621389152268201821459570845676382074318847 4\
9987585020730899986917823591371911555635516591423588955452022783135 6\
71487

fidn = \
5409208666935906335721984381240568833348817180189163039867420502026 0\
3507584668541417769025208694639931728933008322423509307190287576749 6\
1785889362348812903140773300475493807100598065676881503444228877731 1\
9059989476341777895419074498481700768782320951401455481898439487386 1\
8220239172328873908980179133068206058328406706473776470732179720247 4\
5564869254118307907751513551827918816199003714367827297039013650791 2\
9046920472259724063189140599690215913367438566049823637163871029753 6\
0456435278499429021889532061826693207112954237078980237621032647099 3\
9167030108546836799865428209107903256433930575377314268154982589429 0\
22240
```

El cifrado del contador cuando hay un solo archivo infectado sería el siguiente:

```
r1 = \
4428883370782571666620222572687688351079125928689927873344729895990 6\
7448948450658311988935554171166819153389893091726838374040389605219 9\
1904612705406065406965322143854398780752975488365921363804765272182 1\
3044131954833607676960523270278859333005293380951215197976914734253 3\
1641417079738698529518899692215270710548541058645918277024514629067 6\
3577351120757831830656061917197782258237685034384572567170296841619 5\
7940843737260037047731988895646435765832070387010232542795316085705 8\
3347087275722206400008346136639105719376519187987805015121492716423 1\
1215306617572463813675529515688089173045542587996624594986655688886 3\
63293

a1 = \
5409208666935906335721984381240568833348817180189163039867420502026 0\
3507584668541417769025208694639931728933008322423509307190287576749 6\
1785889362348812903140773300475493807100598065676881503444228877731 1\
9059989476341777895419074498481700768782320951401455481898439487386 1\
8220239172328873908980179133068206063000748341719442228719514537461 5\
7774139105957406609014960827128305522852680935119617613152667559521 9\
2911881908785641193316156270329532206511127894303495115845556980437 8\
5231102224010277274161307621389152268201821459570845676382074318847 4\
9987585020730899986917823591371911555635516591423588955452022783135 6\
71488
```

```
24
25 b1 = \
26 162839961341225410563661317900076587330183068871786923775165556716 09\
27 806301572646137162122441956081076404810215200956518440839645166429 95\
28 670194100043511056731147936418286444790237937765261763037137854534 36\
29 511938747351687655566970480929582458319451735971688304192578523001 00\
30 006497130454348380326072297991367588756521408066664788422162275387 76\
31 730387113775380772990405911145501682994853098008464906627693934320 14\
32 217957264771933571949668557871554366507448051272468150913707194152 87\
33 571483921936572545699237541385029159754636304916881656850367220829 32\
34 633094167059895327079485715231903160096949885386249076919100027130 38\
35 123394289578786969155629456473578110929340350814334141567087828835 07\
36 473613816484798726655590011071761189035198843474799102676170732312 61\
37 012132483583299752864196172392738023038484480853681624633289496542 40\
38 588602233542632918327384321993816409192237213338575425955042828053 91\
39 141792582279266063509201259348931071402040192950435836544146037338 24\
40 919810651184690819413572819016886914858285475436995994033286802898 80\
41 787831375096451607829421121180043571573744498777370385518748934832 62\
42 034300940801740550140625908048720845408932082814086218980920335621 06\
43 730348602537565353551928975948323153612140903247792316508632319058 90\
44 6090159077
45
46 c1 = \
47 185986370259077981237181005416794086587971574982718442583126603345 63\
48 926765891647042348584393871344290407090039585613833744047846508018 49\
49 308648348738141832095219133007762620410874403723803737867711479216 28\
50 930155582618694450338029804286590934100566926405403767298522482746 07\
51 089042678764701303267747349625510555851534290152868018747095654001 97\
52 917399462188865659793319881911248515494033543479921097288736484860 51\
53 033914765087518685670698331278042799716904455138590902918045444322 85\
54 091770979896905301638030346439619299372563102267864449035569542223 60\
55 774968805341767275012447236582277464556661042007024196708973629149 67\
56 921524729686636217333832429139468330533187501354040801199361239177 65\
57 396579021696598527453171068379890213113629569878388611812850791569 39\
58 209673336845602115315073796117985936760709586805426268252242121382 80\
59 871243823770660410080704689803982537060770334064504951555259914970 44\
60 088457611157568062798516275689797563655802999721307705879119665386 61\
61 720454008240718199653779649189692999085528552696090514534591343900 89\
62 406471580250832976165025146255969503276911672079553258633225698993 53\
63 905255426473855555308754482261627631491762807385604841822510365897 34\
64 104807308607107241204901026941763783958166328001338472530488845578 66\
65 2997766814
```

El descifrado sería el siguiente:

```
1 m1 = Mod[(PowerMod[c1, fidn, n^2] - 1)/n*PowerMod[fidn, -1, n],  n] =
     1
```

Si el contador aumenta en una unidad, se vuelve a cifrar como sigue:

```
r2 = \
9153785950706662305505811724233849966320071212146297801504913828808 5\
8062965719413534007769753023480835373415862140041803541790509380796 6\
7954555077926012254443645357950242707208653757762082449908038770463 4\
3813300820976117849008961302044072139185687308819196506947704943492 9\
0368251847892295009224225550417254640468159249248134214146321149504 2\
5559076749734884590739020161027636139414220374837517835242078281752 4\
3138970165460340452341584382093804147471211881143158942900781824233 6\
7790839117937458791220215332473384095359918051260558015446734305713 0\
2713092982242850168508203666925754210256849589624752562970019534595 7\
7451

a2 = \
2091327791769305519107006929335115858457600810936499613910876499751 8\
0472302106479475350463457866075044093698639702711490472560478496070 2\
9471025974327726074592903295972387960315108696823457126982851038982 1\
3483724178857012451090891276435994098816821168391192304044664424911 4\
1715882270750542262094224012596535229465471722390712490720290326161 9\
1044118106023505465962338526769953479932139889513772879497790354008 7\
8498722654031037993917281046845312329263608590047136549223836944928 2\
6120580378572380575768231514942094389904898996188472412207718636178 8\
9168434436236392229454087579326517690163721986277993164988472311689 7\
7196551697944854655120354018053585501370346518937474608316346495202 3\
3195442269083983282507521256880192371920602962819781209495308508261 7\
4072141901079044777659514198432338504829346927571709118711947462232 1\
1538854139986879018340250576141486649293034547904344771554770018869 7\
0351226400358700620878312920306640559095658064921795752140932934349 8\
5210973652967455798939864793624990833127716299551850350358958849029 8\
0251117854052143445006291713318954349800788453817361317477024631544 5\
7762099121459705604768830564745344175745935319571234646641003961736 1\
4792660146766491288578730779352044143041917527548846285523453720984 1\
9905374551

b2 = \
2711876778131316706673075488886451090948694693406895888966769049907 9\
5161511260786391303389824206545817788730412062912047334090511071044 0\
7026375520186946142320046841106726471901913594894634967705485124552 7\
5486610009620567512377430186447847126586688623358303248861497597963 5\
9914112253273842034204744090252707154723333564521781931866929125517 7\
7612904167810667978918026897397980643244618913374634136911234961850 2\
8947214019565825135019771714728448049533295892568728642227283434816 3\
3147001743767844348609325203024446123503083986641335888645932154169 3\
4270589323183466328694248685518352460827194773408538551156358721120 1\
6307508056869137624971147185874005061817462918203320419221352025804 9\
3528895428067607325611095300343070717994573914508006730520227611529 0\
0676024931598623311724870806215418584920352432042017556801170832900 2\
7419360769907923820230788534551340068772157397364202656797034157665 8\
0979623926441033644356049076261497870930359886172485710688749019332 0\
4618737891582285726259735776561216079752461143742665058563381692321 5\
5423979255388217329735368877937691933274277195177406479232781078951 5\
```

```
5735964788265306281181976721190492868644377568101312936372511233175\
7855558738272989035078336540381906833531217899686260660008699375356\
8\
0794619371

c2 = \
2683384769602383687325076695948399481127652543505387504206897713053\
5\
6237806364042436918924762186029130243451668767241241940377445947358\
0\
8642256878397981232665075809478566741630815299951598544516560273860\
9\
5880649855480219614237415889543008821948407197436543141918617239227\
1\
9579107910847634077295909681461878614763357673726678960083692826770\
4\
4410232153572612356981374165630174111337660814674870042224982659237\
0\
3342938959345996953957165708271203359835105414208701031187009660989\
6\
9735003210405517097145830479569706574854573301224704627864276026085\
1\
7048639920557458312098155160254851340468667746853533985332275868115\
0\
5493583841160582929950376433528886532237666265737586369348709762714\
9\
0776691316390995698289376968687179512270684438515650001412018167033\
0\
2630200259594447048473973127057280612311998644736224437912998944516\
9222263201346949036630209313456689394077306084148401892754924818011\
0\
2816038194795846848456697963042812607894359537514991772772701666158\
2\
0757978220926077749982811628506728576941424338166582705272248427896\
0\
9898867449552203402344897682483858211605174678765730883355029944364\
9\
9593217855695506916075288364143992611396732926538252784691984846634\
8\
0455631960486612597290203035755953619059671133012754455940257276253\
2\
6369594048
```

Se vuelve a descifrar sin problema:

```
m2 = Mod[(PowerMod[c2, fidn, n^2] - 1)/n*PowerMod[fidn, -1, n], n]= 2
```

Si el contador vuelve a aumentar en una unidad, se procede de igual forma.

```
r3 = \
2532115498248244087253332997554954974444121506804499862566232028583\
6\
1664962990886121278730067009935631550301601087778162112801970731881\
9\
0738585935302700750735617746707663774727735833732709102555943702663\
3\
6601062734480907900266751136983408305479361802331941640957437423643\
0\
1991259627110640701078230879605520532461426449731034194536493371637\
0\
9897275354403559855509577816328130415311875158265962925415995405317\
0\
0683183147743434517736430728106289622992847226884252114401443191031\
7\
7311011493868157789199063988142974690694931160897394998944857214576\
1\
9499137612906377532110882265317058335179724464503286606931445532556\
1\
07290

a3 = \
2190372134268749486904952850054900890389209393833705109885219521572\
4\
4973023557977559143612089206870862184438579717434090720953365203661\
5\
0239239587112462549897978181844645974150626583958938984368281972936\
4\
7066277018306995518503453820187842439473675389904497324106984108723\
8\
2068741974056619054815412567877267091702406505588798054299437548053\
0\
5350974199428090275806263003086113628338111079242231562473279190092\
2\
```

```
728999558945071817244637834923174762452466213370391551294628320867 01\
656053572996171179617021009940196566952590868335904299647928061126 00\
135695320514007972466541407235353322218074996546017607866515942521 05\
844665013480711993596172651470382372313490812581012760774729913204 19\
837426131027746884770906910378956366617761431687700696094562597324 09\
036940717771652877501677479115999950979495911715394943987774884053 23\
254293812667623702678622293992558900548169127611040842624325270348 55\
084961637060046668642535361477909345876755641894738172398483015739 00\
325397179706716283389398178390147339922202622090885222128257827742 31\
763559612231896310991457608555830366746268556845063128025499974142 28\
930967922463170753226763714292485956826699546856979156369665869225 91\
762976254324684906870607086620360588453506632273094403433162434583 64\
7001563024

b3 = \
771418846586528772717324185888915220344280112998225521517672462536 18\
571917800070823985488220864657101505077892982319904211587224350803 23\
463926983265468839862241303277100904217376717962196176390477135650 42\
133185974129157609776647874052315350395928445544232515126996587674 03\
987325457064403294931538832619932998200527392512221239123316148020 33\
775391745277519514919676208588626736852202903366998510435814528834 38\
535377861834337159741557860436738837027330909354112987326828290056 04\
945898114284111254210272740806379130334807434518443178217743621970 07\
118197088424728242133111335680585259659652229616938603276591010077 28\
642422899606925575708367000538735172620316098130237851207235682827 89\
026771673281402585870887255255513535226930442077217871920923722875 47\
134339032228836217012417929980337088953940289329464801224820505250 17\
374843139560417187182388400371835226714118323435748152176299589090 56\
505694954194341856894352469350595869087124113228623843047501893344 87\
601000623508129903685117112902895521848197847415348514307702882353 75\
301142873910979264528049944170439802994322404419568949281092405663 53\
928160292165703537215200876762294916722857860585140064162650382116 02\
190405327120111560118584014801478901610108558701545754213803222099 86\
454238943

c3 = \
307730462305115681486077870387595020858177010130758915899578501080 61\
279369407815439583301361464401328538227972201011004027417521296803 46\
041726984024553549844027111780545150505504087255292324115446309142 14\
210848928069869497836571446398343991341984174029216545202599953433 23\
816838171351806676814816034485357233997330426913549637112857006121 87\
961829196198208212488609992912991978035761527490247148405959536283 90\
909720943906826274047562082010910532737837912787021525114294867369 23\
698694682515826026353153177971302491203109127176552318840155840838 44\
533149621645805141478444361607940501604053438455350079846226935027 40\
145279202217440559905350235596433956738026717817387890760678757173 93\
157149463058246899274059157359608320188681671009763289379898923647 24\
337852796444765824437289225552479368555951335565718498465980727905 36\
682831606818167599020545552440845474081137012979482339335141142931 86\
995194286098287496732206520972476774406536898812326011050244317712 92\
```

```
939278653840131540836323038741182376388523090313780961403660519066 13\
355997491930153874111045153083377925333050260202733478960157980761 65\
904452328921856965967801818513100125936434452151977423577181280736 23\
919423873415258166107270720949565773234275443192747001051033375789 81\
852301550

m3 = Mod[(PowerMod[c3, fidn, n^2] - 1)/n*PowerMod[fidn, -1, n], n] =
   3
```

5 Conclusión

A lo largo del trabajo hemos visto la gran utilidad que tiene la criptografía en el mundo en el que vivimos. Usamos ordenadores y teléfonos móviles para prácticamente todo. Esto pone en riesgo la información que en ellos se pueda encontrar (trabajo, estudios, banca electrónica, información personal, etc). Pero, como hemos visto, los criptosistemas de clave pública son los encargados de protegerla. Estos criptosistemas son muy seguros, pues sus algoritmos de cifrado son muy díficiles de romper. Esto se debe a que usan números tan grandes que no son tratables ni por los ordenadores más rápidos y potentes del mundo.

Sin embargo, en la otra cara de la moneda se encuentran los hackers, quienes también se aprovechan del poder de la criptografía para crear sofisticados malware, e igualmente seguros, con los que robar toda esa información.

Esto lleva a una guerra entre quienes intentan proteger la información y quienes intentan robarla. Pero, ¿qué pasaría si la **computación cuántica** se hiciese realidad? De momento, es solo una teoría, pero se podría crear ordenadores tan potentes que podrían romper los algoritmos de los criptosistemas de clave pública con suma facilidad, incluso trabajando por medio de la fuerza bruta. Es decir, supondrían el fin de la criptografía de clave pública tal y como la conocemos. La diferencia entre esos ordenadores y los actuales es que usarían los que se conocen como **qubits** (quantum bits) en lugar de bits. Los bits operan en binario (0 y 1) y los qubits pueden tomar simultáneamente los valores 0 y 1, lo que permitiría a esos ordenadores realizar distintos cálculos a la vez.

Si esto sucediese, podría provocar un caos, pues las mayores potencias mundiales intentarían hacerse con esos ordenadores y, así, conocer los secretos de las demás. A día de hoy, Google e IBM son los más avanzados en la creación de los ordenadores cuánticos. En septiembre de 2019, Google revelaba que había creado un ordenador cuántico formado por 53 qubits que le permitían hacer en apenas 3 minutos unos cálculos que a un ordenador actual le costaría hacer 10 mil años. Esto se debe a que es capaz de realizar a la vez 2^{53} cálculos. Por otro lado, IBM afirma haber creado otro ordenador, no cuántico, capaz de realizar esos cálculos en tan solo 2 o 3 días.

Esto nos lleva a pensar que, más pronto que tarde, la computación cuántica será realidad, lo que, como hemos dicho, supondrá el fin de la criptografía de clave pública. Entonces, puede que se ideen nuevas formas de cifrar y descifrar información, pudiendo darse una situación parecida a la actual.

Bibliografía

[1] P. Caballero Gil, *Introducción a la Criptografía, 2a edición*, RA-MA Editorial, 2002.

[2] T. ElGamal, *A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms*, IEEE TRANSACTIONS ON INFORMATION THEORY Vol. 31 (4), 1985.

[3] C.C. Elisan, *Advanced Malware Analysis*, McGraw-Hill Education, 2015.

[4] J.L. Gómez Pardo, *Cifrado homomórfico: ejemplos y aplicaciones*, La Gaceta de la RSME, Vol. 15 (4), 2012.

[5] A. Gómez Vieites, *Auditoría de seguridad informática*, RA-MA Editorial, 2014.

[6] A. Menezes, P. Van Oorschot, S.A. Vanstone, *Handbook of Applied Cryptography*, CRC Press, 2001.

[7] C. Paar, J. Pelzl, *Understanding Cryptography*, Springer, 2010.

[8] P. Paillier, *Public-Key Cryptosystems Based on Composite Degree Residuosity Classes, Advances in Cryptology - EUROCRYPT '99, vol. 1592 of Lecture Notes in Computer Science*, Springer-Verlag, 1999.

[9] R.L. Rivest, A. Shamir, L. Adleman, *A Method for Obtaining Digital Signatures and Public-Key Cryptosystems*, Communications of the ACM, Vol. 21 (2), 1978.

[10] J. Saxe, H. Sanders, *Malware Data Science, Attack Detection and Atribution*, No Starch Press, 2018.

[11] N.P. Smart, *Cryptography Made Simple*, Springer, 2016.

[12] B. Scheneier, *Applied Cryptography*, Wiley, 1995.

[13] W. Trappe, L.C. Washington, *Introduction to Cryptography with Coding Theory*, Pearson, 2006.

[14] A. Young, M. Yung, *Malicious Cryptography. Exposing cryptovirology*, Wiley Publishing, Inc., 2004.

[15] https://latam.kaspersky.com/resource-center/threats/riskware

[16] https://softwarelab.org/es/que-es-adware/

[17] https://t-systemsblog.es/backdoor-puerta-trasera/

[18] https://tools.cisco.com/security/center/resources/virus_differences

[19] https://www.adslzone.net/redes/diferentes-tipos-de-/malware-que-abundan-en-la-red/

[20] https://www.avast.com/es-es/c-online-threats

[21] https://www.bbc.co.uk/bitesize/topics/zd92fg8/articles/zcmbgk7

[22] https://www.elevenpaths.com/wp-content/uploads/2020/03/aptualizador-es.pdf

[23] https://www.kaspersky.es/blog/5-weirdest-modern-trojans/8615/

[24] https://www.kaspersky.es/resource-center/threats/ransomware-examples

[25] https://www.malware.es/windows-spyware/coolwebsearch/

[26] https://www.recoverylabs.com/ayuda-y-soporte/data-recovery-white-papers/informes-de-investigacion/virus-chernobyl/

[27] https://www.recoverylabs.com/ayuda-y-soporte/data-recovery-white-papers/informes-de-investigacion/virus-mydoom/

[28] https://www.redeszone.net/2012/12/10/necurs-un-rootkit-/que-ya-ha-infectado-a-mas-de-83-000-equipos/

[29] https://www.redeszone.net/2018/06/16/kits-exploits-2018/

[30] https://www.supformacion.es/no-soy-un-robot/

[31] https://www.techedgegroup.com/es/blog/supremacia-cuantica-google-vs-ibm

[32] https://www.tecon.es/sandboxing-protege-del-malware/

[33] https://www.vilatec.com/tipos-de-malware-y-anti-malware/

[34] https://www.websecurity.digicert.com/security-topics/difference-between-virus-worm-and-trojan-horse

[35] https://www.welivesecurity.com/la-es/2015/04/17/que-es-un-backdoor/

[36] https://www.youtube.com/watch?v=n8mbzU0X2nQ