**Universidad de Costa Rica**
**Facultad de Ingeniería**
**Escuela de Ingeniería Eléctrica**

# Implementación y análisis de algoritmos criptográficos en un FPGA.

**Por:**

**Alejandro León Torres**

**Ciudad Universitaria "Rodrigo Facio", Costa Rica**

**Noviembre de 2015**

# Implementación y análisis de algoritmos criptográficos en un FPGA.

Por:

Alejandro León Torres

**IE-0499 Proyecto eléctrico**

Aprobado por el Tribunal:

M.Sc. Diego Valverde Garro
Profesor guía

M.Sc. Carlos Duarte Martínez
Profesor lector

M.Sc. Enrique Coen Alfaro
Profesor lector

# Dedicatoria

Dedico este proyecto a las siguientes personas:

- First itemtext
- Second itemtext
- Last itemtext
- First itemtext
- Second itemtext

# Reconocimientos

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

# Resumen

El presente proyecto busca investigar sobre la teoría de criptografía y como la misma es implementada en la computación para el resguardo de datos, específicamente en hardware haciendo uso de FPGAs y el lenguaje de descripción de hardware Verilog.

Como punto de partida se elegirán dos algoritmos criptográficos comúnmente empleados, para llevar a cabo un análisis comparativo de una serie de parámetros que son relevantes para su implementación en una plataforma de FPGA. Estos parámetros consideran tres de las mayores limitantes para implementación de algoritmos en FPGA, como son el consumo de celdas, la cantidad de memoria interna utilizada y finalmente el uso de celdas especializadas de aritmética o DSP.

# Índice general

# Índice de figuras

# Índice de tablas

# 1 Introducción

## 1.1 Justificación

El cifrado de datos para aplicaciones de seguridad es de vital importancia en la actualidad. Un ejemplo de esto la fuga de datos de Home Depot en el año 2014, donde se perdió una computadora que contenía información personal de 10000 empleados y clientes causando un costo de millones de dólares a la empresa (Vance, 2008). Otro ejemplo fue la noticia en mayo de 2015 sobre el buscador web *UC Browser* donde indica que es posible robar datos personales por falta de cifrado en la transmisión de datos (Hamilton, 2015). También en febrero de 2015 se dio a conocer la fuga de decenas de millones de datos de la compañía de seguros *Anthem*. Los datos robados (números de seguro, lugar de residencia, números de teléfono, entre otros), se encontraban en una base de datos en la cual los datos se encontraban sin cifrar como indica Yadron y Beck (2015):

> *"Anthem descubrió que hackers irrumpieron en la base de datos y se hicieron de la información de decenas de millones de consumidores, siendo esta la mayor violación de seguridad informática en las compañías de servicios de salud. Debido a que los datos no se encontraban cifrados, eran fácilmente leíbles por los hackers."*

Dada la importancia del cifrado de datos, ahora nos debemos plantear ¿Porqué es importante cifrar datos utilizando hardware?.

En el caso del cifrado implementado en software, trae consigo problemáticas como lo son la necesidad de actualizaciones y el impacto en el desempeño del computador. Este último aspecto se debe a la necesidad de dedicar recursos del sistema para cifrar y descifrar la información (Apricorn, 2015).

En el caso de hardware se tienen los beneficios de que es muy confiable, rápido y conveniente. A diferencia del cifrado en software se tiene una parte de hardware dedicada al proceso de cifrado/descifrado y por tanto el desempeño de la computadora no se ve tan afectado (Apricorn, 2015). Otra de las grandes ventajas del cifrado basado en hardware es la facilidad de configuración, ya que gran parte de esto proceso es eliminado debido a que el hardware se encarga directamente de esto (SANS, 2007).

Debido a lo expuesto anteriormente se buscó trabajar en algoritmos criptográficos en hardware ya que es un área de trabajo que se puede explotar para investigar y mejorar.

Se eligió como plataforma de trabajo un FPGA debido al bajo costo que permite usarla como un plataforma de desarrollo conveniente, la popularidad que ha alcanzado en los últimos años y las mejoras que se le han realizado para que estos dispositivos logren desempeños similares a los ASICs.

## 1.2 Alcances y limitaciones del proyecto

Se implementarán dos algoritmos criptográficos en un FPGA haciendo uso de Verilog como lenguaje de descripción de hardware.

Posteriomente y mediante las herramientas de síntesis de Xilinx se realizará un análisis de métricas críticas en el desarrollo de aplicaciones en FPGAs como los son la cantidad de compuertas o celdas, el consumo de memoria interna del FPGA así como la cantidad de bloques aritméticos o de DSP que son usados por cada algoritmo. Uno de los factores que aunque tiene importancia no se tomará en cuenta es el consumo de potencia.

Inicialmente se va llevará a cabo un análisis individual de cada algoritmo, haciendo uso de las métricas anteriormente descritas y variando parámetros comunes de los algoritmos criptográficos como lo son el tamaño de la llave y la cantidad de rondas de cifrado (este último en algoritmos de tipo Feistel).

Como segunda parte del proyecto se realizará un análisis comparativo entre ambos algoritmos eligiendo parámetros fijos para ambos.

Los análisis individuales y comparativos anteriomente mencionados no abarcarán ningún tipo de criptoanálisis de algún algoritmo con respecto otro ni de cuál sería la escogencia de los parámetros ideal para realizar un análisis comparativo entre ambos. Sino que a partir del análisis individual realizado se va a efectuar una escogencia de los parámetros de ambos algoritmos para realizar su comparación implementando las métricas descritas.

Para la escogencia de estos dos algoritmos se realizará a partir de una identificación de las principales ramas del cifrado para así elegir los dos algoritmos de dos de estas ramas abarcando de esta manera un tema más amplio para el análisis y discusión.

Se limitará a realizar una escogencia de esta manera donde también tendrá mucho peso que los algoritmos que sean implementables, de manera relativamente sencilla, en un FPGA conociendo desde un principio las limitantes estáticas del hardware.

## 1.3 Objetivos

### Objetivo General

Implementar dos algoritmos de cifrado en un FPGA y realizar un análisis comparativo de la implementación de ambos algoritmos, empleando una serie de parámetros previamente seleccionados.

### Objetivos Específicos

1. Implementar dos algoritmos criptográficos comúnmente empleados en un FPGA utilizando el lenguaje de descripción de hardware Verilog.
2. Realizar un análisis individual para cada uno de los algoritmos individuales, variando alguno de sus parámetros (por ejemplo el tamaño de la llave) para comparar haciendo uso de métricas como la cantidad de compuertas, el consumo de memoria interna del FPGA así como la cantidad de bloques aritméticos o de DSP que se van a ir utilizando conforme se varíe el parámetro del algoritmo elegido.
3. Realizar un análisis comparativo de los dos algoritmos implementados, utilizando como métricas la cantidad de compuertas o celdas, el consumo de memoria interna del FPGA así como la cantidad de bloques aritméticos o de DSP que son usados por cada algoritmo.

## 1.4 Metodología

La metodología que se siguió para la realización del proyecto es la siguiente:

1. Estudios bibliográficos de:
    - Criptografía: importancia y como la misma se implementa en la computación.
    - Algoritmos criptográficos: Identificación de ramas y subramas.
    - Código e implementación de algoritmos criptográficos en diferentes lenguajes de programación.
2. Escogencia de los algoritmos criptográficos a implementar.
3. Implementación de los algoritmos criptográficos en un FPGA de Xilinx.
4. Realización del análisis individual de cada uno de los algoritmos.
5. Realización del análisis comparativo entre ambos algoritmos.
6. Realización de las conclusiones y Recomendaciones.

## 1.5 Desarrollo

El presente informe se estructura para el lector de la siguiente manera:

1. Capítulo I: Introducción.
2. Capítulo II: Antecedentes y Marco Teórico.
3. Capítulo III: Implementación de los algoritmos criptográficos en el FPGA.
4. Capítulo IV: Resultados de los análisis individuales y comparativos de los 2 algoritmos implementados en el FPGA.
5. Capítulo V: Conclusiones y recomendaciones.

# 2 Marco Teórico

Este capítulo va a poner en contexto al lector con respecto a conceptos básicos de criptografía, definir y explicar los algoritmos que van a ser implementados en el FPGA, y realizar una breve explicación sobre características de un FPGA para explicar el porqué de la escogencia de los parámetros que van a ser analizadas.

## 2.1 Conceptos Básicos

Según la RAE (2015a) la criptografía se define como

> *Arte de escribir con clave secreta o de un modo enigmático.*

El mensaje que se desea transmitir es llamado *texto plano* o simplemente *mensaje*. Este mensaje pasa por un proceso donde se disfraza el texto plano en un *texto cifrado*, el proceso se denomina *cifrado*[1]. El proceso inverso donde se toma un texto cifrado en un texto plano se denomina *descifrado* (Schneier, 1996).

El texto plano o mensaje se denota por la letra M o P, el texto cifrado se denota por la letra C, la función o algoritmo que cifra se denota por E y la que descrifa se denota por D. Un algoritmo criptográfico corresponde a la función matemática para cifrar y descrifrar.
Se muestra en las Ecuaciones (2.1) y (2.2) las relaciones entre estas notaciones. Note como al aplicarle la función de cifrado al texto plano se obtiene el texto cifrado y como al aplicarle la función de descifrado al texto cifrado se obtiene el texto plano. Finalmente se debe cumplir la identidad que describe la Ecuación (2.3) (Schneier, 1996).

$$E(M) = C \tag{2.1}$$

$$D(C) = M \tag{2.2}$$

$$D(E(M)) = M \tag{2.3}$$

La importancia de la criptografía transciende más allá de brindar la confidencialidad en la comunicación, la criptografía también cumple con la siguientes tareas:

[1]Según el ISO 7498-2 los términos correctos para encriptar y desencriptar son cifrar y descifrar respectivamente.

- Autenticación: El receptor del mensaje debe de poder conocer y asegurar el emisor del mensaje, esto para que un tercero no pueda adjudicarse la identidad del emisor.

- Integridad: El receptor tiene que poder asegurarse que el mensaje no fue cambiado en el transito del mismo. Esto para que un tercero no pueda cambiar el contenido enviado por el emisor sin que el receptor lo sepa.

- *Non-repudiation*: El emisor del mensaje no puede negar que el mensaje fue enviado por él.

## 2.2 Sistema criptográfico (*criptosistema*)

Cuando la seguridad del algoritmo se basa en como procede el algoritmo, se denomina *algoritmo restringido*. Este tipo de algoritmos son poco utilizados en la actualidad debido al gran problema que presentan. Tomemos de ejemplo que un grupo de usuarios decide utilizar un algoritmo criptográfico restringido para sus comunicaciones, se tendrá una comunicación segura hasta que alguno de los miembros decida salirse del grupo, ya que el usuario al no pertenecer más al grupo, no le importa mantener en secreto el algoritmo y puede distribuirlo para que terceros intercepten las comunicaciones. Así cada vez que un miembro deja el grupo, se debe proceder a cambiarse a todo un nuevo algoritmo lo cual puede tornarse una labor complicada. También ocurre el problema que si se logra obtener un algoritmo equivalente para cifrar los datos, se debe migrar a un nuevo algoritmo.

En cambio, la criptografía moderna (Denning, 1982) agrega el concepto de *llave* donde se tiene un algoritmo el cual toma como parámetros de entrada una llave y el texto plano o cifrado, y cifra o descifra el mismo de forma correcta, únicamente si se tiene la llave correcta. El esquema descrito anteriormente se muestra en la Figura 2.1. Retomando el ejemplo anterior, el grupo solamente necesitaría cambiar de llave cuando un miembro se va, facilitando el uso del cifrado y manteniendo las comunicaciones secretas.

Estos conceptos viene a definir lo que actualmente se conoce como sistema criptográfico o *criptosistema*. Según Denning (1982), un criptosistema cuenta con 5 componentes:

- Un espacio de textos planos o mensajes ($M$)

- Un espacio de textos cifrados ($C$)

- Un espacio de llaves ($k$)

- Una familia de *transformaciones de cifrado*: $E_K : M \to C$ donde $K \epsilon k$

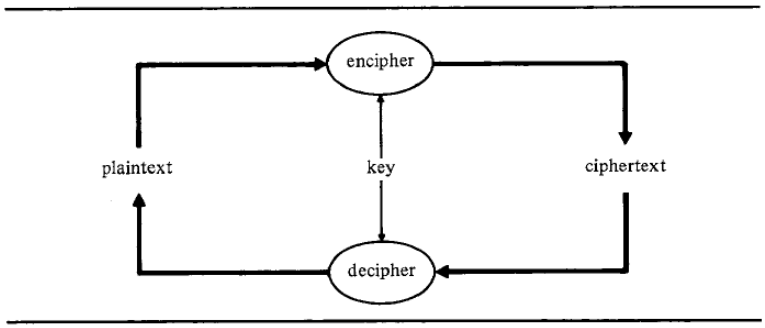


Figura 2.1: Descripción gráfica de un sistema criptográfico (Denning, 1982).

- Una familia de *transformaciones de descrifrado*: $C_K : C \rightarrow M$ donde $K \epsilon k$

Se entiende como *espacio* el conjunto de posibles valores para la variable dada, sea esta M, C o K. Y una familia de transformaciones corresponde a todos los posibles mapeos que se pueden realizar de un espacio a otro (de $M$ a $C$ o viceversa) con todos los valores contenidos en el espacio $k$.

En la actualidad se trabaja la criptografía sobre computadoras, es decir, cifrando y descrifrando bits, los cuales pueden tener diferentes significados, ya sea una imagen, un texto, un programa, etc. Esto significa que en la actualidad no se trabaja sobre caracteres del alfabeto o símbolos, sino más bien sobre 1's y 0's. Esto toma relevancia ya que al tener solo dos símbolos para cifrar, los algoritmos se vuelven más complejos por la falta de alternativas para sustituir un símbolo por otro.

Antiguamente la criptografía se basaba en caracteres que eran sustituidos o traspuestos por otros caracteres. Esto corresponde a cifrados de sustitución y de transposición, los cuales continúan siendo la base de la criptografía pero basado en los dos símbolos del sistema binario.

Como se explicó anteriormente el cifrado de sustitución se basa en tomar un caracter del texto plano y sustituirlo por otro caracter. Para descrifrar el texto cifrado simplemente se sustituyen de vuelta los caracteres y listo.

Según Schneier (1996), en la criptografía clásica existen 4 tipos de cifrado por sustitución:

- Cifrado de sustitución simple: Una sustitución de uno a uno entre cada caracter del texto plano y el texto cifrado. Ejemplos de este tipo de sustitución son el famoso cifrado de César y el ROT13 utilizado en UNIX.

- Cifrado de sustitución homofónico: Una sustición de uno a muchos. Un caracter del texto plano, por ejemplo A, puede ser sustituido por varios

| Letter | Homophones |
|---|---|
| A | 17 19 34 41 56 60 67 83 |
| I | 08 22 53 65 88 90 |
| L | 03 44 76 |
| N | 02 09 15 27 32 40 59 |
| O | 01 11 23 28 42 54 70 80 |
| P | 33 91 |
| T | 05 10 20 29 45 58 64 78 99 |

One possible encipherment of the message is:

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| $M$ = | P | L | A | I | N | P | I | L | O | T |
| $C$ = | 91 | 44 | 56 | 65 | 59 | 33 | 08 | 76 | 28 | 78 |

Figura 2.2: Ejemplo de cifrado homofónico (Denning, 1982).

caracteres en el texto crifado, por ejemplo “5”, “13”, “43”. Observe la Figura 2.2 donde se presenta una serie de posibles asignaciones de números a las letras del mensaje PLAIN PILOT y un posible texto cifrado haciendo uso de este tipo de cifrado.

- Cifrado de sustitución de poligrama: Una sustitución por bloques en donde se toma un bloque de caracteres del texto plano y se sustituye por su bloque equivalente en el texto cifrado. Por ejemplo si en el texto plano se tiene “ABC” se sustituye por “SLL” en el texto cifrado.

La otra variedad de algoritmos son los de transposición, en este tipo de algoritmos criptográficos el texto plano se convierte en texto cifrado cuando el orden de los caracteres es cambiado bajo alguna norma. Un ejemplo moderno de este tipo de algoritmos es el *rail-fence* donde el texto plano se reacomoda con la forma de una cerca como se observa en la Figura 2.3. En este caso la llave del algoritmo sería la profundidad de la cerca, para efectos de este ejemplo es de 3.

Actualmente en los algoritmos que se implementan en computadoras se combina tanto la transposición como la sustitución. Por ejemplo se tiene el algoritmo RC5 (el cual se desarrollará más adelante en este proyecto), en donde se utiliza corrimientos o rotaciones a bits (transposición) y sumas o XOR's (sustituciones) para cifrar el texto plano. Los algoritmos que se implementan en la criptografía moderna se dividen en dos categorías principales: simétricos y de llave pública (también llamados asimétricos (Denning, 1982)).

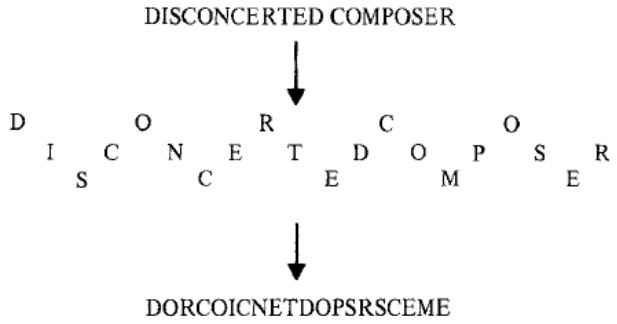


Figura 2.3: Ejemplo de cifrado de transposición (Denning, 1982).

## 2.3 Algoritmos criptográficos

### Algoritmos simétricos

Schneier (1996) da una muy buena analogía para explicar el concepto de un algoritmo simétrico. Piense en el algoritmo como una caja fuerte. La combinación de la caja fuerte vendría a ser la *llave* del algoritmo. Note como en una caja fuerte cualquier persona con la combinación puede llegar, abrir la caja y poner o sacar documentos de la misma. En el caso de no conocer la combinación, se debe proceder a forzar la caja o probando todas las combinaciones posibles hasta hallar la correcta. Es decir en un algoritmo simétrico se cuenta con una llave única que funciona tanto para cifrar como para descrifrar los mensajes como se muestra en la Figura 2.4. La notación para el cifrado y descrifado en estos algoritmos se muestra en las Ecuaciones (2.4) y (2.5).

$$E_K(M) = C \tag{2.4}$$

$$D_K(C) = M \tag{2.5}$$

Los algoritmos simétricos se dividen en 2 categorías (Schneier, 1996):

- Cifrado de bloque: Se cifra en bloques de bits ya sea *bytes, words*, etc. Es decir cuando se va a proceder a cifrar un texto plano, se segmenta el texto en grupos de bits y estos son cifrados en conjunto. Se puede tomar como ejemplo el algoritmo *Data Encryption Standart* (DES) el cual cifra sobre bloques de 64 bits. Otros ejemplos son:
  - Lucifer.
  - LOKI.

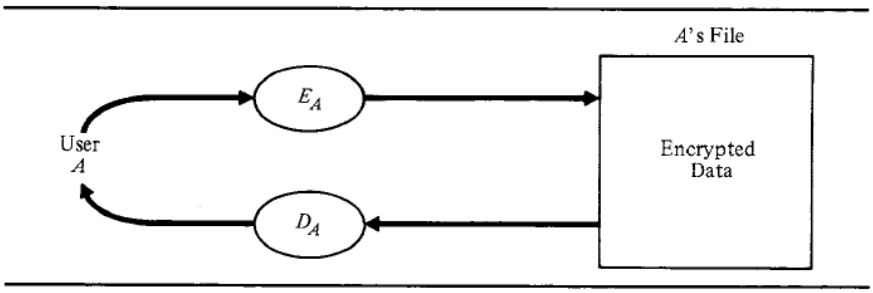


Figura 2.4: Descripción gráfica de una algoritmo de llave simétrica (Denning, 1982).

  - 3-way.
  - RC5.
  - GOST.
  - IDEA.

- Cifrado de *Stream*: Es cuando se trabaja sobre un bit únicamente. Estos no son muy usuales en la actualidad ya que al trabajar en lenguaje binario solo se cuenta con 2 símbolos y si se cifra únicamente un bit no existen muchas posibilidades para sustituir o transposicionar. Ejemplos de estos algoritmos pueden ser:

  - RC4.
  - SEAL.
  - WAKE.

Según Schneier (1996), los criptosistemas simétricos en la red afrontan los siguientes problemas

- Distribución de la llave: La llave se debe mantener en secreto. Esto en la actualidad es una tarea demasiado díficil de lograr porque la llave debe ser conocida para el cifrado y descifrado (emisores y receptores) entonces para establecer una comunicación segura el primer paso debe ser entregar la llave de forma segura, lo cual en una red de computadoras se puede tornar una tarea prácticamente imposible de realizar. La única solución sería entregar las llaves mediantes un servicio de *courier* o similares e igualmente se corren riesgos.

- Compromiso de seguridad: Si la llave es conocida por un tercero, si este intercepta el tráfico de información, todas las comunicaciones serán descifradas fácilmente.

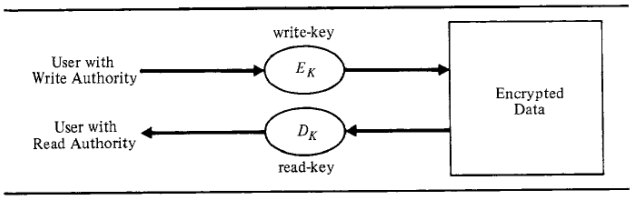


Figura 2.5: Descripción gráfica de una algoritmo de llave pública (Denning, 1982).

- Comunicaciones aisladas: En el caso de que cada usuario de una red se desee comunicar secretamente por separado con todos los otros usuarios haciendo uso del mismo criptosistema, se debe utilizar una llave diferente para cada comunicación. Así para una red de N usuarios se requieren $N(N-1)/2$ llaves. A primera vista esto no parece tan importante por ejemplo para 10 usuarios, se necesitan 45 llaves lo cual está bien, pero para 100 usuarios se necesitarían 4950 llaves que se tienen que distribuir de forma secreta.

### Algoritmos de llave pública

Nuevamente Schneier (1996), nos ofrece una excelente analogía para explicar, en este caso, los algoritmos de llave pública. Tenemos un *mailbox*, donde cualquier persona puede poner un mensaje adentro pero **únicamente** el dueño puede abrirlo para sacar y leer los mensajes.

En los algoritmos de llave pública los emisores hacen uso de una llave para cifrar los mensajes que desean enviar pero estos mensajes pueden ser descrifrados únicamente si se tiene la llave para descifrar mensajes (que es diferente de la llave para cifrar), la cual el receptor la tendrá bien resguardada. En la Figura 2.5 se muestra el diagrama básico de un algoritmo de llave pública. Ejemplos de estos algoritmos pueden ser:

- RSA
- Pohlig-Hellman
- Rabin
- ElGamal

Estos algoritmos sientan su base matemática en funciones llamadas *one-way* en donde al aplicarle una función a una variable, la variable no puede

retornar a su valor original de ninguna manera, es decir la función no tiene una inversa.

Según Schneier (1996), existen funciones *one-way* con una "puerta trasera" donde se puede retornar a la variable original conociendo ciertos parámetros, este tipo de funciones son las que se implementan en algoritmos de llave pública. Los algoritmos dominantes de esta rama se basan en la dificultad de factorizar números grandes que son el resultado de multiplicar dos números primos grandes como también se basan en el *Discrete Logarithm Problem*.

En estos algoritmos no es posible a partir de la llave para cifrar obtener la llave para descrifrar. Esto permite que la llave para cifrar se pueda hacer pública, por lo cual recibe el nombre de llave pública y la llave para descrifrar se denomina llave privada. La notación para estos algoritmos corresponde a la de las Ecuaciones (2.6) y (2.7)

$$E_{K_x}(M) = C \tag{2.6}$$

$$D_{K_y}(C) = M \tag{2.7}$$

El objetivo de utilizar cifrado de llave pública se basa en:

- Receptor y emisor acuerdan un sistema criptográfico.
- El receptor entrega al emisor su llave pública.
- El emisor cifra el texto plano haciendo uso de la llave pública entregada y el sistema acordado.
- El emisor envía el texto cifrado.
- El receptor descrifa el texto cifrado haciendo uso de la llave privada.

De esta manera no hay forma en que fisgones logren descrifrar el mensaje aunque obtengan la llave pública. Así el receptor se asegura que las comunicaciones van a ser mucho más seguras, ya que el emisor deja de tener la llave para descifrar y no puede brindarsela a nadie o que le sea robada a este. Para la implementación de un criptosistema que hace uso de algoritmos de llave pública, lo que se hace es que en la red se tiene una base de datos donde se registra el usuario y su respectiva llave pública. Así cuando un usuario desea comunicarse con otro, va a la base de datos, busca al usuario y su llave pública, cifra el mensaje con la misma y se la envía. Esto solventa los problemas que presentaba anteriormente los algoritmos simétricos, ya que no es necesario transmitir de forma secreta llaves para poder realizar comunicaciones y para que diferentes usuarios se comuniquen de forma secreta entre si no es necesario el uso de diferentes llaves por cada enlace de comunicación.

Algunos usos de estos algoritmos en la actualidad son:

- Llave maestra para el sistema de pago digital de un banco.
- La clave que utiliza un gobierno para certificar sus visas o pasaportes.
- La firma digital de un notario público.

### Criptosistemas híbridos

Los beneficios que conlleva utilizar algoritmos de llave pública son grandes pero se pagan a un precio muy alto: tiempo de procesamiento. Según Schneier (1996), el tiempo de procesamiento del RSA con respecto al del DES es de alrededor de 1000 mil veces más lento.

En un mundo donde la velocidad es una clave fundamental en las comunicaciones se propuso la siguiente solución. Ya que los algoritmos simétricos tienen la debilidad de comunicar la llave antes de comenzar la comunicación pero son mucho más rápidos, y que los algoritmos de llave pública ostentan una mejor sistema para comunicarse en la red secretamente pero son muy lentos, se decidió utilizar ambos. La llave del algoritmo simétrico es cifrado con un algoritmo de llave pública para transmistirse en la red sin comprometerla y posterior a esto se realizan las comunicaciones con el algoritmo simétrico para obtener una mejor velocidad de comunicación. Se puede ver el protocolo de la siguiente manera (Schneier, 1996):

- El receptor envía su llave pública al emisor.
- El emisor genera una llave de sesión[2], la cifra y se la envía al receptor usando la llave pública que le fue dada.
- El receptor descrifra la llave de sesión usando su llave privada.
- Ahora ambos pueden comunicarse de forma secreta con la llave de sesión con un criptosistema simétrico.

## 2.4 Seguridad en algoritmos criptográficos

Como parte de la investigación previa para este proyecto no podemos dejar de lado la otra cara de la moneda, el criptoanálisis. La ciencia que abarca la criptografía y el criptoanálisis es conocida como criptología (Denning, 1982). Claramente el objetivo de la criptografía es mantener un mensaje en secreto de

[2]Una llave de sesión se utiliza en comunicaciones donde la idea es cifrar cada comunicación de manera individual con una llave distinta. Son útiles cuando la llave se crea al inicio de la comunicación y se destruye al final de la misma (Schneier, 1996).

terceros, pues el criptoanálisis según RAE (2015b), es el "arte de descifrar criptogramas", así es como los terceros buscan inmiscuirse en las comunicaciones secretas sin tener un acceso a la llave.

Según Schneier (1996), el criptógrafo al diseñar su algoritmo debe asumir que el criptoanalista puede tener un acceso completo a las comunicaciones entre emisor y receptor y al algoritmo que implementa el criptosistema, así toda la seguridad del criptosistema debe residir en la llave.

La acción en la que un criptoanalista atenta contra un criptosistema es denominada *ataque*. Según Schneier (1996); Denning (1982), los principales tipos de ataques son:

- *Ciphertext-only*: El criptoanalista debe obtener la llave a partir de varios textos cifrados. Debe conocer el tema tratado de las comunicaciones que está interviniendo (lo cual es obvio, ya que sino para qué está interviniendo las comunicaciones) entonces puede saber de forma previa que ciertas cosas que puede contener el texto plano.

- *Known-plaintext*: El criptoanalista tiene acceso a ciertos textos planos y sus respectivos textos cifrados. A partir de estos debe deducir la llave o debe generar un algoritmo de descifrado equivalente al diseñado por el criptográfo.

- *Chosen-plaintext*: El criptoanalista puede obtener el texto cifrado de un texto plano que él seleccione, esto es mucho más poderoso porque puede cifrar mensajes que den más información acerca de la llave. Un sistema de bases de datos es vulnerable a este tipo de ataques debido a que los usuarios pueden agregar elementos a la base de datos y ver el resultado del texto cifrado en la misma. Nuevamente su tarea es obtener la llave o generar un algoritmo de descifrado equivalente.

- *Chosen-ciphertext*: Este tipo de ataque se da en algoritmos de llave pública, en donde el criptoanalista tiene acceso al texto cifrado que va a ser descifrado y al texto plano. Es decir tiene una "caja negra" donde la entrada es el texto cifrado y la salida el texto plano. El objetivo es obtener la llave a partir de estos recursos.

- *Chosen-key*: El criptoanalista tiene conocimiento sobre la relación entre las diferentes llaves.

- *Rubber-hose*: El criptoanalista amenaza, extorsiona, chantajea u obtiene de alguna forma la llave sin ningún tiempo.

Otro ejemplo es que los algoritmos de llave pública son vulnerables a ataques de tipo *chosen-plaintext*. Tome por ejemplo M como un texto plano del

espacio $M$ de posibles mensajes. La tarea de un criptoanalista es tomar todos M$\epsilon M$ y cifrarlos para obtener todos los posibles C$\epsilon C$. De esta manera solo debe interceptar las comunicaciones que desea y comparar sus valores de texto cifrado con los que intercepta para descifrar las comunicaciones. Note que el criptoanalista no pudo obtener la llave pero si logró descrifrar las comunicaciones.

### Tamaño de la llave

Como se mencionó anteriormente la seguridad de un buen algoritmo depende de su llave. Asumiendo que se tiene una seguridad del algoritmo perfecta, la única forma de quebrar el criptosistema sería mediante un ataque de fuerza bruta, el cual es un tipo de ataque *known-plaintext*, para obtener la llave.

Analizando el tamaño de la llave, si se tuviera una de $2^8$ bits, probando las 256 posibilidades se obtendría la llave, lo cual equivale a unos cuantos segundos de procesamiento en una computadora. Para $2^{64}$ bits tenemos $1{,}8446744^{19}$ posibles llaves, lo cual tomaría con los recursos computacionales de una supercomputadora alrededor de 585,000 años. Para una llave de 2048, con un billón de intentos por segundo en computadoras en paralelo se necesitarían 10597 años para encontrar la llave (Schneier, 1996).

Entonces ¿Porqué no hacer un algoritmo con una llave muy grande?. La respuesta es porque conforme aumenta el tamaño de la llave, se paga en tiempo de procesamiento. Así se debe obtener un balance donde la llave sea lo suficientemente grande para que sea el algoritmo sea seguro, pero lo suficientemente pequeña para que el tiempo de procesamiento sea bueno (Schneier, 1996).

Comparar el nivel de seguridad ante un ataque de fuerza bruta de un algoritmo simétrico y uno de llave pública con llaves del mismo tamaño no es posible, pero la Figura 2.6 muestra una tabla con equivalentes realizados empíricamente por Schneier (1996) sobre tamaños de llaves que dan un nivel de seguridad equivalente para los dos tipos de algoritmos, esto nos indica que no es recomendable comparar algoritmos de distintas ramas entre sí.

### Manejo de las llaves

Se puede tener un algoritmo extremadamente robusto, veloz y con un tamaño de llave perfecto, pero si un atacante puede obtener la llave por algún medio el algoritmo es completamente inútil. De ahí la gran importancia de como manejar las llaves.

Al ser esto tan importante existen ciertas maneras de generar llaves:

- Ingresada por el usuario: El usuario elige una llave y esa es la que se va a utilizar. Este método es sumamente inseguro debido a que gran

| Symmetric Key Length | Public-key Key Length |
|---|---|
| 56 bits | 384 bits |
| 64 bits | 512 bits |
| 80 bits | 768 bits |
| 112 bits | 1792 bits |
| 128 bits | 2304 bits |

Figura 2.6: Comparación del tamaño de llaves en algoritmos simétricos y asimétricos (Schneier, 1996).

cantidad de contraseñas se repiten, o son datos personales del usuario como su número de celular o similares. Por tanto se pueden realizar *ataques de diccionario* donde las primeras llaves que se prueban son las mencionadas anteriormente.

- *Random keys*: Es un muy buen método para generar llaves, consiste en utilizar un programa que genere la llave, es ventajoso ya que es robusto ante ataques de diccionario pero presenta el problema que la llave es difícil de recordar y posiblemente se olvide.

- *Pass phrases*: Es una combinación de ambos, el usuario escribe una contraseña fácil de recordar y después hace uso de una función *one-way* el sistema convierte esta llave en una llave random de tamaño arbitrario.

## 2.5 *Field Programmable Gate Array* (FPGA)

Un FPGA consiste en una matriz de bloques lógicos configurables, conectados mediantes interconexiones programables. Posterior al proceso de manufactura el FPGA puede ser reprogramado para darle una nueva funcionalidad, esto

Tabla 2.1: Ventajas y beneficios de diseñar con FPGAs (Xilinx, 2015c).

| Ventaja | Beneficio |
|---|---|
| Tiempo de producción más corto | Pasos de manufactura como layout, máscara u otros no son necesarios |
| No upfront non-recurring expenses (NRE) | Costo que ocurre en el diseño de ASICs. |
| Ciclo de diseño más simple | El software se encarga en gran parte del *placing, routing* y *timing.* |
| Ciclo del proyecto más predecible | Debido a que se eliminan los posibles re-spins, capacidades de la oblea, etc. |
| *Field reprogramability* | Se puede cargar un nuevo RTL y listo. |

Tabla 2.2: Ventajas y beneficios de diseñar con ASICs (Xilinx, 2015c).

| Ventaja | Beneficio |
|---|---|
| 100 % personalizados | Ya que se manufactura apegado a la especificación. |
| Costos unitarios más bajos | Para diseños de gran volumen. |
| Más pequeños | Debido a que el diseño se acoge completamente a la especificación. |

es lo que diferencia a un FPGA de un *Application Specific Integrated Circuits* (ASIC) (Xilinx, 2015b).

### FPGA vs ASIC

En años anteriores los FPGAs presentaban gran desventaja en cuanto a velocidad, consumo y área respecto a un ASIC y por eso aunque eran reprogramables, desarrollar aplicaciones en un FPGA solo se daba para realizar prototipos y no para un producto final. En la actualidad los FPGAs se encuentran en auge debido a que su proceso de manufactura ha mejorado mucho en los aspectos anteriormente descritos, por ejemplo un FPGA ya puede alcanzar los 500MHz, han aumentado la densidad de lógica y albergan otras funcionalidades como por ejemplo módulos de *Digital Signal Proccessing* (DSP), de tiempos y hasta de *high-speed serial* (Xilinx, 2015b).

Estos grandes avances en conjunto con un flujo de diseño mucho más fácil, barato y rápido, indican que el desarrollo en FPGAs va a convertirse en la nueva forma de diseñar hardware. La Tabla 2.1 muestra beneficios y ventajas del diseño con FPGAs y la Figura 2.7 muestra la diferencia en el ciclo de diseño de hardware con FPGAs y ASICs. Aún con todas estas ventajas los FPGAs todavía presentan ciertas desventajas con respecto a los ASICs, como se muestra en la Tabla 2.2.

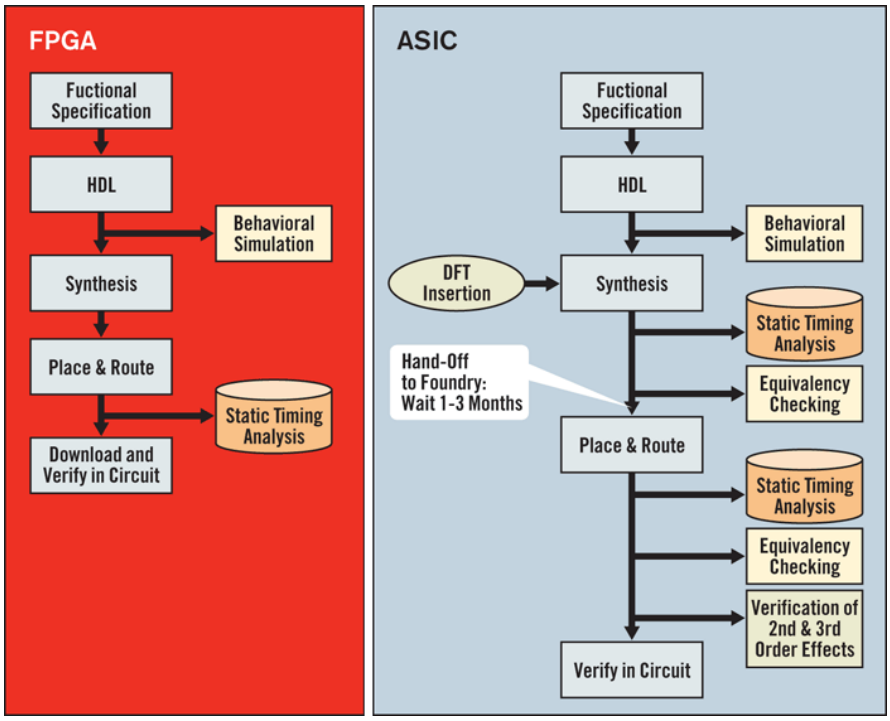


Figura 2.7: Ciclo de diseño de hardware para FPGAs y ASICs (Xilinx, 2015c).

### Componentes básicos de un FPGA

Todos los FPGAs, como se muestra en la Figura 2.8, tiene como mínimo 3 componentes (Xilinx, 2011)

- *Configurable Logic Block*
- *Programmable Switching Matrix*
- *Input/Output Block*

A parte de estos componentes un FGPA también puede tener un módulos de memoria RAM, ROM y hasta módulos para DSP.

A continuación se describen de forma general los componentes que conforman los bloques descritos anteriormente (Xilinx, 2011, 2015a)

- *Configurable Logic Block* (CLB): Corresponde a la unidad principal de diseño para lógica combinatoria y flip-flops. Se encuentra compuesto por slices y la cantidad que lo componen varía según el modelo. Para determinar la densidad de slices por CLB se debe sintetizar el diseño previamente para decidir cual es el modelo que se ajusta al mejor desempeño

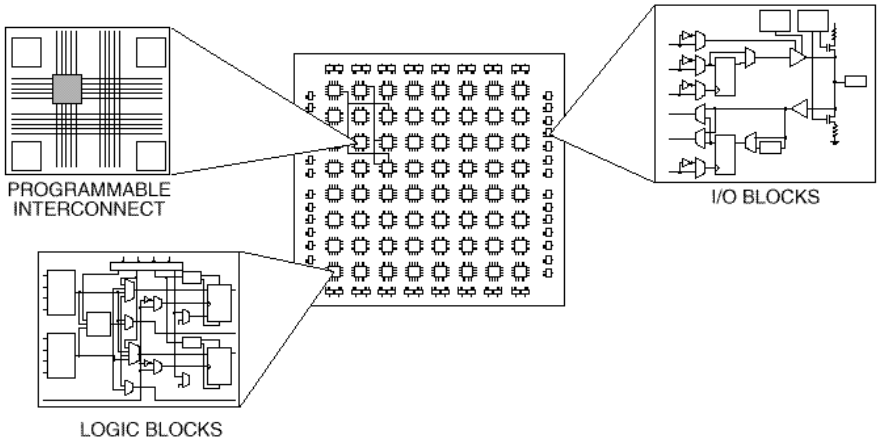


Figura 2.8: Estructura básica de un FPGA. (Egyetem, 2015)

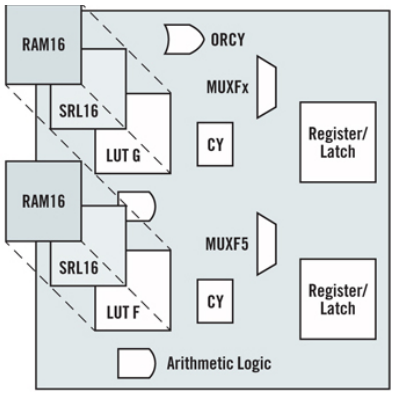


Figura 2.9: Estructura básica de un CLB (Xilinx, 2015d).

del diseño. Estos bloques son conectados a un PSM para comunicarse con otras partes del FPGA y también se conectan a un IOB para comunicarse con el mundo exterior. La Figura 2.9 muestra una representación general de lo que puede contener un CLB.

- *Slice*: Es la unidad básica de un FPGA, puede estar compuesto por:
  - *Look Up Table* (LUT): También son llamados Function Generators, depende del número de entradas y no de la complejidad de la función que se está implementando. Así el retardo es constante para todo el LUT. Son usados para implementar lógica booleano y también como memoria o SRL[3]

[3]Shift Register Lookup Table (SRL) es un registro desplazable de tamaño variable. Sus

  - *Wide Multiplexers*: Se utilizan para conectar LUTs entre si y también para no utilizar LUTs en operaciones simples, ahorrando LUTs y mejorando la velocidad de procesamiento.
  - *Carry Chain*: Cuando los CLBs deben de realizar operaciones como sumas o restas utilizan esta característica para mejorar la velocidad en la que se realiza la operación
  - *Registers*: Pueden ser flip-flops o latches.

  Ciertos slices pueden se utilizados como una memoria RAM distribuida, en donde por ejemplo el LUT se utiliza como una memoria, ya sea en configuración *single, dual, simple-dual o quad port.*

- *Programmable Switch Matrix* (PSM): Corresponde a un módulo en donde se realiza la interconexión de los distintos CLBs y otros componentes del FPGA según requiera el diseño. Esta interconexión puede realizarse de forma vertical, horizontal y diagonal.

- Input/Output Blocks (IOB): Estos bloques se encargan de comunicar al FPGA con el mundo exterior, manejan gran cantidad de estándares de comunicación como por ejemplo: LVCMOS, PCI, I2C y SSTL.

Como se puede observar los FPGAs son de gran importancia y por eso el objetivo de implementar los algoritmos criptográficos en un FPGA adquiere relevancia. Así también es muy importante conocer que tantos recursos (slices, memoria y DSP) se requieren para implementar estos diseños en un FPGA ya que al aumentar la cantidad de recursos aumentan los costos de producción, volviendo este proyecto relevante por su proyección innovadora sin dejar de lado el aspecto económico tan importante en la industria.

## 2.6 Escogencia de los algoritmos a implementar

En un principio el objetivo de este proyecto era realizar una comparación con las métricas descritas en la Introducción sobre un algoritmo simétrico y un algoritmo de llave pública, posterior a la realización del marco teórico se llegó a la conclusión de que realizar este análisis no iba a ser para nada justo debido a tres razones:

- El tiempo de procesamiento: Como se menciona anteriormente los algoritmos de llave pública consumen mucho más tiempo realizando el cifrado y descifrado que un algoritmo de llave simétrica.

usos pueden ser como retardo programable o como una memoria FIFO.

- La formas en las que estos algoritmos son diseñados producen falencias en seguridad distintas, como se indica en la sección 2.4 los tamaños de llaves varían mucho para lograr una seguridad similar ante un ataque de fuerza bruta, esto haría poco justo realizar un análisis comparativo con el mismo tamaño de llave.

- Los algoritmos realizan el proceso de cifrado de maneras muy diferentes: En el caso de los algoritmos simétricos son basados en transposición y sustitución que trabajan en su mayoría con XOR's, sumas y corrimientos. En cambio los algoritmos de llave pública trabajan basados en la multiplicación de números primos y *Discrete Logarithm Problem* donde ocurren muchas multiplicaciones y sumas. Esto produce un consumo de recursos muy diferente entre ambos tipos.

Bajo el criterio que ambos algoritmos tuvieran características similares para poder ser analizados comparativamente y que fueran fáciles de implementar a nivel de hardware se optó por comparar dos algoritmos simétricos, específicamente el RC5 y el FEAL los cuales están basados en redes de Feistel[4] y por tanto comparten características de diseño para que la comparación sea un poco más justa.

## 2.7 Descripción de los algoritmos a implementar

### RC5

El algoritmo criptográfico de llave simétrica RC5 fue creado en 1997 por Ronald Rivest para la *RSA Data Security*. Según **?** los objetivos de este algoritmo se basaban en que fuera adaptable tanto para hardware como para software, que tuviera una llave de tamaño variable, que el tamaño de la palabra a cifrar fuera de tamaño variable para que así se ajustara a procesadores de tamaños de palabra diferentes y por último que el número de rondas de cifrado fuera variable (con la finalidad de permitir decidir que tanta seguridad y que tan rápido se cifran los datos).

#### Terminología del algoritmo

Por simplicidad como se tienen tantos parámetros variables para el algoritmo entonces a la hora de llamarlo se utiliza la siguiente notación: RC5-w/r/b.

[4] Las redes Feistel toman un bloque de texto plano de tamaño N (par) y lo dividen en dos mitades denominadas L y R. El cifrado en este tipo de redes es iterativo donde el resultado de la iteración $i$ depende del resultado de la iteración $i-1$. Ejemplos de algoritmos de este tipo pueden ser DES, Lucifer, FEAL, Khufu, Khafre, LOKI, GOST, CAST, Blowfish, entre otros.

Estas y otras variables se definen a continuación:

- w: es el tamaño de la palabra, el valor nominal es de 32 y los permitidos son 16,32 y 64. Y el algoritmo cifra en bloques de tamaño 2w. Además cada palabra contiene $u = w/8$ bytes.
- r: corresponde al número de rondas, los cuales pueden variar entre 0 y 255.
- S: tabla con la expansión de la llave. Tiene un tamaño de $t = 2(r + 1)$ palabras.
- llave: Se tiene dos parámetros para definirla:
  - b: cantidad de bytes que tiene la llave.
  - K[i]: K-ésimo byte de la llave va desde K[0] hasta K[b-1].
- A y B: Corresponden a las partes izquierda y derecha del texto plano respectivamente.
- L: Arreglo de $c = max(b, 1)/u$ palabras que funciona para convertir la llave de bytes a palabras. Inicialmente se encuentra lleno de ceros.
- $P_w$ y $Q_w$: Constantes que se definen como:

$$P_w = odd(2^w(e-2)); e = \text{número de euler} \tag{2.8}$$

$$Q_w = odd(2^w(\phi-2)); \phi = \text{golden ratio} \tag{2.9}$$

Según **?** los beneficios de utilizar el RC5 como algoritmo de cifrado se tiene una serie de parámetros variables los cuales conforme pasa el tiempo se pueden adaptar a la necesidades de cifrado. Un ejemplo de lo necesario que es esto es el problema con el algoritmo DES , para cuando fue creado su tamaño de llave era ideal pero ahora es muy corta y no hay forma fácil de que el algoritmo acepte una más fácil.

Otros usos del algoritmo es por ejemplo (**?**) RC5-32/8/0, es decir si el uso de una llave, para generar una secuencia de números pseudo-aleatorios.

También es importante destacar que:

- + indica suma complemento a dos. Así como - indica resta en complemento a dos.
- $\oplus$ indica un XOR bit a bit.
- <<< indica rotación hacia la izquierda, igualmente >>> indica rotación a la derecha. Entonces X <<< Y significa que la palabra X se rota Y bits a la izquierda.

### Pasos del algoritmo

A continuación se muestra el pseudo-código extraído de **?** para cifrar y descifrar datos con este algoritmo.

- Expansión de llave: Corresponde la primer paso antes de poder cifrar o descifrar texto plano. Recibe como entrada una llave de tamaño arbitrario y realiza lo siguiente:

```
1  for i = b - 1 downto 0 do
2    L [i/u] = (L[i/u] <<< 8) + K[i]
3
4  S[0] = Pw;
5  for i = 1 to t-1 do
6    S[i] = S[i-1] + Qw;
7
8  i = j =0
9  A = B =0
10 do 3*max(t,c) times:
11   A = S[i] = (S[i] + A + B) <<< 3
12   B = L[j] = (L[j] + A + B) <<< ( A + B )
13   i = (i+1) mod(t)
14   j = (j+1) mod(c)
```

## IDEA o LUCIFER o FEAL

# Bibliografía


Apricorn (2015). Software vs hardware encryption. Recuperado de `http://www.apricorn.com/software-vs-hardware-encryption/`. [Consulta 1 set. 2015].

Denning, D. (1982). *Cryptography and Data Security*. Addison Wesley Publishing Company, Inc.

Egyetem, M. (2015). Field programmable gate arrays (fpga). Recuperado de `http://mazsola.iit.uni-miskolc.hu/cae/docs/pld1.en.html`. [Consulta 12 oct. 2015].

Hamilton, D. (2015). Five data leak nightmares. Recuperado de `http://www.thewhir.com/web-hosting-news/uc-browsers-lack-of-encryption-could-have-leaked-personal-data-to-spies`. [Consulta 16 oct. 2015].

RAE (2015a). Definición criptoanálisis. Recuperado de `http://buscon.rae.es/drae/srv/search?val=criptoan\%E1lisis`. [Consulta 10 oct de 2015.].

RAE (2015b). Definición criptografía. Recuperado de `http://buscon.rae.es/drae/srv/search?val=criptograf\%EDas`. [Consulta 10 oct de 2015.].

SANS (2007). Hardware versus software a usability comparison of software-based encryption with seagate drivetrust hardware-based encryption. Recuperado de `http://www.seagate.com/staticfiles/SeagateCryptofaceoff.pdf`. [Consulta 1 set. 2015].

Schneier, B. (1996). *Applied Cryptography, Second Edition: Protocols, Algorthms, and Source Code in C*. John Wiley & Sons, Inc.

Vance, J. (2008). Five data leak nightmares. Recuperado de `http://www.networkworld.com/article/2289232/lan-wan/five-data-leak-nightmares.html`. [Consulta 16 oct de 2015.].

Xilinx (2011). Basic fpga architectures. Recuperado de `http://web.mit.edu/clarkds/www/Files/slides2.pdf`. [Consulta 12 oct. 2015].

Xilinx (2015a). 7-series clb architecture overview. Recuperado de `http://www.xilinx.com/training/fpga/7_series_CLB_architecture_video.htm`. [Consulta 12 oct. 2015].

Xilinx (2015b). Field programmable gate array (fpga). Recuperado de `http://www.xilinx.com/training/fpga/fpga-field-programmable-gate-array.htm`. [Consulta 12 oct. 2015].

Xilinx (2015c). Fpga vs. asic. Recuperado de `http://www.xilinx.com/fpga/asic.htm`. [Consulta 12 oct. 2015].

Xilinx (2015d). What is a fpga? Recuperado de `http://www.xilinx.com/fpga/`. [Consulta 12 oct. 2015].

Yadron, D. y Beck, M. (2015). Health insurer anthem didn't encrypt data in theft. Recuperado de `http://www.wsj.com/articles/investigators-eye-china-in-anthem-hack-1423167560`. [Consulta 16 oct. 2015].