UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

DESARROLLO DE FRAMEWORK PARA INTERACCIÓN CON LLAVES ASIMÉTRICAS EN APLICACIONES WEB

TESIS PROFESIONAL

PARA OBTENER EL TÍTULO DE
INGENIERO EN COMPUTACIÓN

PRESENTA:
GUILLERMO ROMERO GALLEGOS

DIRECTOR DE TESIS
MTRO. JUAN JOSÉ CARREÓN GRANADOS

CIUDAD UNIVERSITARIA SEPTIEMBRE 2015

Agradecimientos

A mi mamá, gracias a su inmenso apoyo tanto económico como afectivo ha llegado al fin la conclusión de esta tesis. Durante la carrera agradezco su gran esfuerzo por ayudarme en todos los aspectos necesarios para acudir a la facultad, por brindarme sabiduría invaluable en momentos difíciles y por supuesto, por nunca dudar un poco de mí.

A mi padre, que indirectamente me ayudó a ser fuerte y obtener la seguridad necesaria para obtener lo que quiero en mi vida.

A mis amigos, son pocos los buenos amigos y se cuentan con los dedos aquellos que siempre estás dispuesto a ayudar y porque no, que te ayuden. No diré nombres porque yo sé que estarán presentes, pero aún no han abierto la cuenta.

Objetivos de la tesis

Para el desarrollo de la tesis se plantean 4 objetivos principales los cuales se enumeran a continuación.

1. Ofrecer una descripción documental de la estructura actual del proyecto así como de los módulos que lo integran con el fin de poder modificar, agregar funcionalidad así como implementar nueva funcionalidad dentro del framework.
2. La implementación de un framework a partir de módulos para interactuar con llaves asimétricas en aplicaciones.
3. La implementación de interfaces para integrar los módulos desarrollados con la intención de poder agregarlos en sistemas de cómputo web para autenticación, inicio de sesiones seguras utilizando elementos de hardware.
4. El desarrollo de un prototipo funcional que integre los módulos y que sirva como marco de referencia para los posteriores desarrollos que aporten funcionalidad al framework.

Índice

Introducción

La integración de la tecnología en la vida diaria se ha incrementado gradualmente, debido al mayor uso de dispositivos accesibles como es el caso de unidades de almacenaje de datos (memorias USB) y teléfonos inteligentes.

A medida que estas tecnologías las integramos gradualmente en nuestras vidas se hace más cotidiano su uso, y nos facilitan algunas tareas.

En ciertos aspectos la tecnología aún no se encuentra establecida para su funcionamiento habitual y se siguen utilizando papeles para todo tipo de tramites. Este es el caso de la firma de documentos donde aún se realiza de manera autógrafa, esto presenta inconvenientes y no garantiza la autenticidad de la misma.

Por otro lado la comunicación confiable entre una persona y una computadora aún se realiza por mecanismos de autenticación simple como es el caso de un usuario y una contraseña, esta contraseña no garantiza la identidad de la persona.

Es en esta línea de acción donde se desarrolló un framework para la interacción con sistemas de cómputo de forma segura a partir de tokens compuestos de llaves asimétricas.

La utilización de mecanismos para desarrollo ágil hace de esta tarea una integración más simple en los sistemas de cómputo actual, aunado con los actuales estándares de comunicación que permiten un mayor rango de acción no solo entre sistemas sino también entre dispositivos.

Con ayuda del framework la firma de documentos digitales se realiza de una manera transparente y simple para el usuario. De una manera similar se utilizan los tokens para comenzar una sesión segura en un sistema de cómputo.

Al integrar el framework se puede establecer todo un sistema de autenticación, firma y validación de documentos de una forma ágil dejando la implementación de estas tareas de lado y prestando más atención en la implementación de las reglas de negocio.

Los resultados obtenidos de la implementación de este framework incluyen la comunicación con dispositivos USB de almacenamiento masivo, la interacción con llaves asimétricas, así como una serie de implementaciones de patrones de diseño enfocados al desarrollo de software.

Por último también se establecen estándares de comunicación segura así como implementaciones para compatibilidad con tecnologías existentes de llaves asimétricas.

Capítulo 1

Criptografía

1.1 Definición

Comencemos por definir que es criptografía, proviene del griego criptos – oculto y grafé – escritura por lo tanto se podría definir como escritura oculta.

El objetivo principal de la criptografía consiste en permitir a dos personas intercambiar información confidencial incluso si nunca se han conocido y solo pueden comunicarse por un canal que está siendo monitoreado por una tercera persona.

Ahora el campo de la criptografía engloba mucho más que la comunicación secreta, incluye autenticación de mensaje, firmas digitales, protocolos para intercambio de llaves, protocolos de autenticación, subastas y elecciones electrónicas así como dinero digital.

Pero de qué herramientas matemáticas se ayuda la criptografía para cumplir con sus objetivos. Algunas de las áreas de las matemáticas incluyen teoría de números, álgebra abstracta (grupos, anillos, campos), probabilidad, estadística y teoría de la información.

1.2 Cifrado por sustitución simple

Para comprender el funcionamiento básico de este tipo de cifrado basta con ubicar la primera letra del mensaje claro y ascender cinco posiciones en el abecedario. Al realizar varios corrimientos encontraremos que la letra V no tiene su equivalente de sustitución es por este motivo que el abecedario se retroalimentara dejando como resultado un circulo. A partir de este círculo de abecedario se puede crear otro círculo dentro con el mismo abecedario y para poder formar un mensaje cifrado es necesario definir un corrimiento N, esto se realiza rotando el abecedario como se ilustra en la Figura 1.1.

La rueda de cifrado puede realizarse con diferentes alfabetos y diferentes corrimientos en diferentes partes del mensaje.

Pero al realizar un ataque para tratar de descifrar el mensaje solo bastaría con realizar 26 posibles combinaciones para obtener el mensaje en claro. Para robustecer este método se puede establecer una tabla de cifrado.

Para realizar una tabla de cifrado se escribe el alfabeto en claro en el renglón superior, en el renglón inferior se puede seleccionar cualquier otro alfabeto con diferentes caracteres, el orden de los caracteres debe ser aleatorio. Un ejemplo de tabla de sustitución con 26 letras del alfabeto se muestra en la Tabla 1.1.

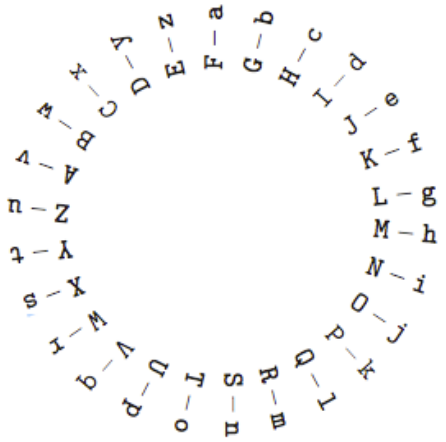


Figura 1.1: Cifrado por sustitución simple con corrimiento de cinco letras

Tabla 1.1: Cifrado por sustitución simple mediante tabla

a	**b**	**c**	**d**	**e**	**f**	**g**	**h**	**i**	**j**	**k**	**l**	**m**	**n**	**o**	**p**	**q**	**r**	**s**	**t**	**u**	**v**	**w**	**x**	**y**	**z**
C	I	S	Q	V	N	F	O	W	A	X	M	T	G	U	H	P	B	K	L	R	E	Y	D	Z	J

1.3 Criptoanálisis para cifrado simple por sustitución

Al realizar el análisis de un texto cifrado por sustitución simple se debe tener en cuenta las posibles combinaciones existentes en un alfabeto. Por lo tanto si se parte de un alfabeto de 26 letras, la primera letra tendrá 26 posibles combinaciones. Para la segunda letra 25 posibles combinaciones por lo tanto.

26 * 25 * … * 2 * 1 = 26! = 4.032914611 X 10^26

Por lo que existen más de 10^26 diferentes tablas de sustitución simple, también conocidas como llaves de cifrado.

Para comenzar a realizar un criptoanálisis se parte del análisis del leguaje ya que la estructura en la construcción de palabras no es aleatoria. Con base en esta suposición se puede definir una tabla de frecuencia de letras. Tal como lo muestra la tabla 1.2.

A partir de un análisis estadístico se puede definir la frecuencia de las letras que más se presentan en un texto. Esta frecuencia varía a razón del lenguaje utilizado.

Otro método es utilizar los bigramas que existen en el lenguaje, se presentan en pares de letras consecutivas en un texto. Un ejemplo de los bigramas más utilizados en el idioma inglés se muestra en la tabla 1.3.

A partir de estos dos métodos se pueden sustituir las letras en el texto de acuerdo a la frecuencia, analizando la estructura de las palabras se puede realizar una combinación entre las frecuencias y definir las posibles soluciones al mensaje cifrado.

Tabla 1.2: Frecuencia de letras en un texto en inglés

Por frecuencia descendente			
E	13.11%	M	2.54%
T	10.47%	U	2.46%
A	8.15%	G	1.99%
O	8.00%	Y	1.98%
N	7.10%	P	1.98%
R	6.83%	W	1.54%
I	6.35%	B	1.44%
S	6.10%	V	0.92%
H	5.26%	K	0.42%
D	3.79%	X	0.17%
L	3.39%	J	0.13%
F	2.92%	Q	0.12%
C	2.76%	Z	0.08%

Tabla 1.3: Bigramas más comunes en inglés (frecuencia cada 1000 palabras)

th	he	an	re	er	in	on	at	nd	st	es	en	of	te	ed
168	132	92	91	88	86	71	68	61	53	52	51	49	46	46

1.4 Conceptos básicos de cifrado

1.4.1 Divisibilidad y máximo común divisor

La teoría de números se basa en el estudio de los números enteros $\mathbb{Z}$.

... , -5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5, ...

Los numero enteros pueden ser sumados, restados y multiplicados de forma usual, y cumplen con la reglas de la aritmética (conmutativa, asociativa y distributiva).

Si *a* y *b* son enteros y se requiere sumar *a* + *b*, restar *a* - *b*, y multiplicar *a* * *b*. En cada caso el resultado es un entero. Pero al trasladar esta definición al dividir dos enteros no siempre se obtiene como resultado un entero.

Definición. Si a y b son enteros con $b \neq 0$. Podemos decir que b divide a, o a es divisible por b. Si c es un entero tal que

$$\frac{a}{b} = c\,;\ a = bc$$

Por lo tanto se puede definir que b divide a y se denota de la siguiente forma $b|a$.

Proposiciones:

Si $a|b$ y $b|c$ entonces $a|c$.

Si $a|b$ y $b|a$ entonces $a = \pm b$.

Si $a|b$ y $a|c$ entonces $a|(b+c)$ y $a|(b-c)$.

Definición. El común divisor de dos números enteros a y b es un entero positivo d que divide a ambos. El máximo común divisor de a y b es el mayor entero positivo d tal que $d|a$ y $d|b$. El mayor común divisor de a y b se denota gcd(a,b).

Si a y b son enteros positivos.
Entonces a dividido por b con un cociente q y un residuo r.

$$a = b \cdot q + r \qquad cuando \qquad 0 \leq r < b$$

Por lo tanto si d es el máximo común divisor de a y b, entonces d es el máximo entero posible que cumple con:

$$a = dx$$
$$b = dy$$

con *x* y *y* enteros positivos.

Para encontrar el máximo común divisor de *a* y *b* se puede utilizar el algoritmo Euclidiano, y solucionando la ecuación para encontrar *x* y *y* tal que

$$ax + by = d$$

1.4.2 Aritmética modular

La aritmética modular también conocida informalmente como aritmética de reloj, define el comportamiento cíclico en un intervalo denominado módulo.

Definición. Si $m \geq 1$ es un entero, podemos decir que los enteros *a* y *b* son congruentes al módulo *m* si la diferencia $a - b$ es divisible por *m*. Se puede escribir de la siguiente forma:

$$a \equiv b \pmod{m}$$

Proposición. Tenemos $m \geq 1$, *m* es entero.
Si $a_1 \equiv a_2 (\mathrm{mod}\, m)$ y $b_1 \equiv b_2 (\mathrm{mod}\, m)$ entonces

$$a_1 \pm b_1 \equiv a_2 \pm b_2 \pmod{m} \text{ y } a_1 \cdot b_1 \equiv a_2 \cdot b_2 \pmod{m}$$

Si *a* es un entero, entonces
$a \bullet b \equiv 1 (\mathrm{mod}\, m)$ para cualquier entero *b* si y solo si gcd(a,m) = 1.

1.4.3 Aritmética modular y cifrado por desplazamiento

Ahora bien si tomamos como ejemplo el cifrado por desplazamiento y a cada letra del alfabeto se le asigna un número podemos definir lo siguiente, se tiene un desplazamiento *k* el cual toma una letra del texto plano correspondiente la cual lleva asociado un numero *p* y asignada a la letra del texto cifrado correspondiente al número p + k mod 26.

En este caso el módulo 26 corresponde al número de letras en el alfabeto.
El desplazamiento sirve de llave para el cifrado y el descifrado.

Por lo tanto se pueden definir una fórmula para el cifrado
(Letra del texto cifrado) ≡ (Letra del texto claro) + (Llave)(mod 26)

y de forma similar para el descifrado
(Letra del texto claro) ≡ (Letra del texto cifrado) - (Llave)(mod 26)

1.5 Esquemas de codificación

Al emplear cualquier tipo de cifrado se refiere no solo a mensajes de texto plano sino también a datos contenidos en una computadora. Estos datos se definen de forma binaria y se toman como cadenas de ocho bits.

Las computadoras utilizan el código ASCII para almacenar información, con base en la asignación de caracteres a números hexadecimales o binarios con el fin de tener un equivalente del alfabeto en caracteres comprensibles por la computadora.

1.6 Pseudoaleatoriedad

El concepto de pseudoaleatoriedad tiene como antecesor a la aleatoriedad y es aquí donde se encuentra su definición, ya que los números aleatorios se generan a partir de una variable definida al azar en función de una distribución uniforme.

Pero al definir el concepto de azar en una computadora no es simple, ya que ésta solo procesa las instrucciones que se le proporcionen de una manera ordenada y predecible. Por tal motivo se utilizan procedimientos los cuales producen la impresión de generar números aleatorios, por lo que los números generados no son completamente aleatorios y se les denomina pseudoaleatorios.

Por más increíble que parezca la generación de números aleatorios no se genera con procedimientos al azar, se ha demostrado que este tipo de procedimientos tiene un comportamiento cíclico y por lo tanto predecible.

Una buena forma de generar números aleatorios en computadora es mediante generadores de congruencia lineal, este tipo de métodos dan la ilusión de aleatoriedad. Una función de aleatoriedad propuesta por Derrick Lehmer dice, es necesario un módulo *m*, un multiplicador *a,* un incremento *c* y un valor de inicio x_0.

$$x_{n+1} = f(x_n) = (ax_n + c) \ mod \ m$$

Es necesario seleccionar los valores apropiados para a, c y m, este último definiendo la periodicidad en la generación de los números aleatorios. Tal como se muestra en la figura 1.2.

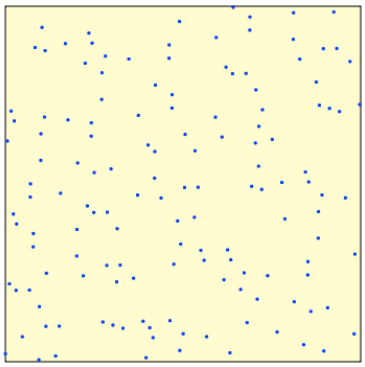


Figura 1.2: Aleatoriedad para $x_{n+1} = (106x_n+64) \ mod \ 8505$

Sin embargo para la generación de llaves criptográficamente seguras se definen generadores estadísticos de números aleatorios, los cuales emplean elementos no predecibles como insumo para generar las llaves. Algunas de estas técnicas utilizan los relojes de las computadoras debido a su exactitud así como a los seriales de los dispositivos internos (hardware) ya que no siempre son conocidas estas propiedades. También se puede utilizar el ruido existente en un dispositivo de entrada como un micrófono o la salida de video.

1.7 Principio de Kerckhoff

Para la definición de un sistema criptográfico seguro se definen algunas propiedades deseables que debe cumplir un algoritmo de cifrado.

Estos principios indican que la seguridad del algoritmo de cifrado únicamente debe radicar en la secrecía de la llave y no en la secrecía del algoritmo.

Si (*K, M, C, e, d*) pertenecen a un tipo de cifrado aceptable deben de seguir las siguientes propiedades:

1. Para cualquier llave *k* que exista en *K* y un mensaje *m* que exista dentro de *M* debe ser fácil calcular el cifrado *c*.
2. Para cualquier llave *k* que exista dentro de *K* y un mensaje cifrado *c* que exista dentro de *C* debe ser fácil calcular el mensaje en claro *d*.
3. Dado uno o más mensajes cifrados $c_1, c_2, \ldots, c_n$ existentes en *C* usando la misma llave *k* existente dentro de *K* debe ser muy difícil calcular cualquiera de sus correspondientes mensajes en claro m_n sin conocer *k*.

1.8 Cifrado simétrico (Llave privada)

En el cifrado simétrico se comparte el conocimiento de la llave secreta k. Usando esta única llave emisor y receptor pueden cifrar y descifrar el mensaje enviado. Al tener solo una llave en común se dice que tiene conocimiento en común o simétrico.

Al definir el cifrado simétrico en forma matemática tenemos una llave *k* seleccionada de un espacio de llaves *K* para cifrar un mensaje en claro *m* seleccionado de un espacio de posibles mensajes *M*, el resultado del proceso de cifrado es un mensaje cifrado *c* de un espacio de posibles mensajes cifrados *C*.

La definición del cifrado puede quedar descrito por la función

$$e : K \times M \rightarrow C$$

cuyo dominio $K \times M$ es el conjunto de pares (k, m) cuyo rango es el espacio de C.
De manera similar se encuentra la función para descifrar

$$d : K \times C \rightarrow M$$

1.8.1 Data Encryption Standard (DES)

El algoritmo DES o también referenciado como DEA (Data Encryption Algorithm) es un cifrado por bloques ya que la entrada de información se descompone en una serie de bloques de 64 bits y usa una llave *K* de 64 bits donde 8 bits son de paridad.
En la figura 1.3 se muestra una descripción gráfica del algoritmo.

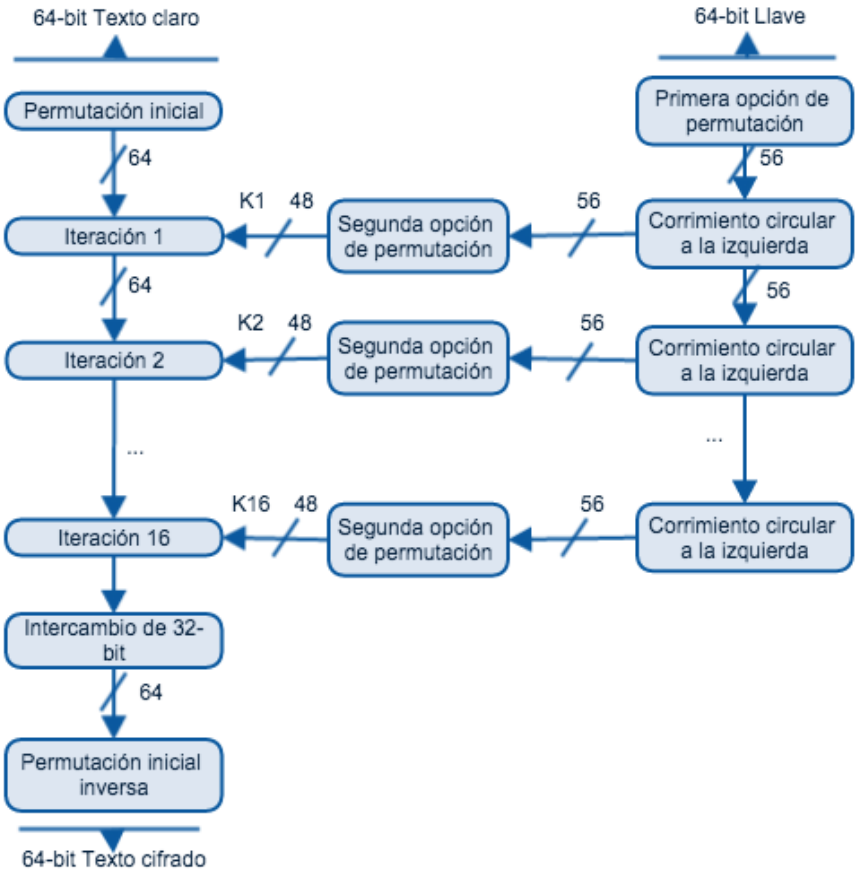


Figura 1.3: Descripción general del algoritmo de cifrado DES

El proceso de cifrado del DES se basa en el cifrado de Feistel el cual genera 16 iteraciones para desordenar la información, es por este motivo que para descifrar la información solo basta seguir el algoritmo DES en reversa para encontrar el mensaje en claro. Figura 1.4 muestra los insumos utilizados por el algoritmo DES.

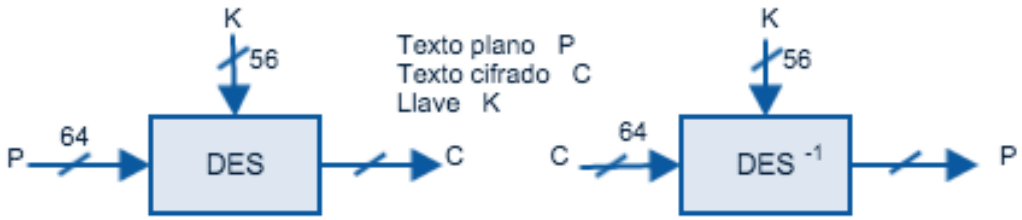


Figura 1.4: DES entrada – salida

Para la inicialización y cierre del algoritmo se utilizan dos tablas denominadas Permutación Inicial (IP) y Permutación Inicial Inversa (IP^{-1}) de 64 bits con números del 1 al 64. Cada entrada en la tabla de permutación indica la posición de una entrada numerada de bit en una salida.

El desarrollo de una iteración comienza por segmentar el mensaje de 64 bits en dos bloques de 32 bits *L* y *R*. Los cuales siguen las siguientes formulas.

$$L_i = R_{i-1}$$
$$R_i = L_{i-1} \oplus F(R_{i-1}, K_i)$$

Como *R* es de 32 bits se utiliza la tabla *E* de expansión para obtener 48 bits y poder realizar el XOR con una iteración de la llave. Posteriormente se realiza una sustitución con la tabla *S-box* para comprimir a 32 bits y posteriormente una permutación P. Por último se realiza un XOR con L_{i-1} para obtener R_i, como lo muestra la figura 1.5.

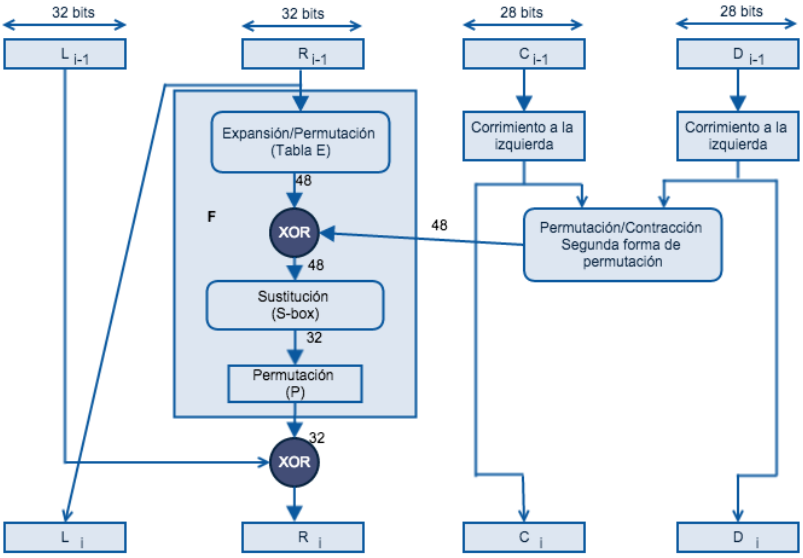


Figura 1.5: Iteración del algoritmo DES

1.8.2 Advanced Encryption Standard (AES)

Rijndael fue con el nombre que se dio a conocer este algoritmo, después de su estandarización por el Instituto Nacional de Estándares y Tecnología (NIST) fue nombrado AES.

El cifrado toma un bloque del mensaje en claro de 128 bits (16 bytes). La longitud de la llave puede ser de 128, 192 y 256 bits por lo que se pueden nombrar AES-128, AES-192 o AES-256.

Los bloques de 16 bytes están representados en una matriz de 4 X 4 bytes, este se considera el estado inicial para cifrar o descifrar la información.

El proceso de cifrado y descifrado utiliza los bloques enteros de información en cada ronda para realizar sustituciones y permutaciones.

La llave proporcionada es expandida a un arreglo lineal de longitud 44 utilizando palabras de 32 bits, cuatro distintas palabras de 128 bits son utilizadas por cada ronda.

Cuatro diferentes etapas son utilizadas en cada ronda

- Substitute bytes: Utiliza la tabla S-box para realizar una sustitución byte por byte del bloque.
- ShiftRows: Realiza una permutación de los renglones 2, 3 y 4.
- MixColumns: Realiza una sustitución utilizando cada columna individualmente. Cada byte de la columna es mapeado en un nuevo valor de la función $GF(2^8)$. Cada valor de la columna está definido por

$$s'_{0,j} = (2 \cdot s_{0,j}) \oplus (3 \cdot s_{1,j}) \oplus s_{2,j} \oplus s_{3,j}$$

$$s'_{1,j} = s_{0,j} \oplus (2 \cdot s_{1,j}) \oplus (3 \cdot s_{2,j}) \oplus s_{3,j}$$

$$s'_{2,j} = s_{0,j} \oplus s_{1,j} \oplus (2 \cdot s_{2,j}) \oplus (3 \cdot s_{3,j})$$

$$s'_{3,j} = (3 \cdot s_{0,j}) \oplus s_{1,j} \oplus s_{2,j} \oplus (2 \cdot s_{3,j})$$

- AddRoundKey: Realiza un XOR bit a bit del bloque actual con la porción de la llave extendida.

Para el cifrado y el descifrado se comienza con AddRoundKey, seguida de las rondas las cuales dependen del tamaño de la llave. 10 rondas para una llave de 16 bytes de longitud, 12 rondas para 24 bytes y 14 rondas para 32 bytes. La figura 1.6 muestra el proceso gráfico para el cifrado y descifrado de en AES..

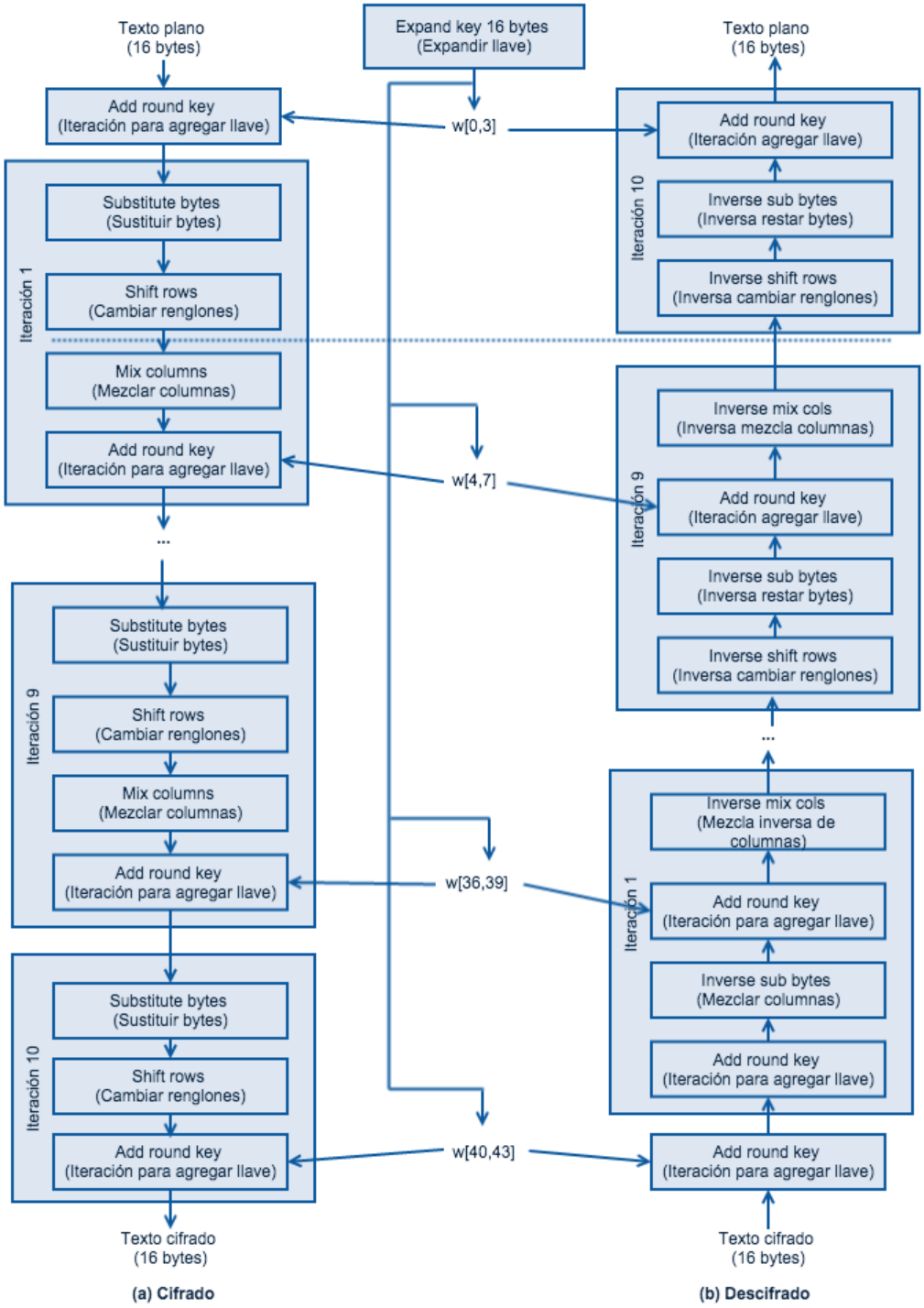


Figura 1.6: AES Cifrado y Descifrado

1.9 Cifrado asimétrico (Llave pública)

Se conoce como cifrado asimétrico debido a que utiliza un juego de llaves (pública y privada) para cifrar y descifrar los datos que son intercambiados entre dos medios.

Pero si es necesario intercambiar información en un medio inseguro utilizando un mecanismo de cifrado simétrico, se debe acordar una llave en común antes de iniciar la comunicación. A partir de este razonamiento se creó un mecanismo para intercambio de llaves públicas el cual se denominó Diffie – Hellman basado en el problema del logaritmo discreto.

1.9.1 Diffie - Hellman

Definición. Sea *g* una raíz primitiva para un campo finito, y *h* un elemento distinto de cero de un campo finito. El problema del logaritmo discreto (Discrete Logarithm Problem DLP) es el problema de buscar un exponente *x* tal que

$$g^x \equiv h \pmod{p}$$

El número *x* es llamado el logaritmo discreto de *h* con base *g* y es denotado como $\log_g(h)$.

Por lo tanto $\log_g(h)$ es definido solo hasta agregar o substraer múltiplos de $p-1$. En otras palabras, $\log_g(h)$ es realmente definido por el módulo de $p-1$.

Para concretar, x se encuentra entre 0 y $p-2$ satisfaciendo la congruencia $g^x \equiv h(\mathrm{mod}\, p)$.

1.9.2 Intercambio de llaves por Diffie - Hellman

Ya que se conoce el problema del logaritmo discreto y la dificultad de calcular su logaritmo, Diffie – Hellman utiliza dicho mecanismo para intercambio de llaves.

La forma general de Diffie - Hellman se define como

$$N \equiv g^{secret\ int} \pmod{p}$$

p es un gran número primo y g es un número distinto de cero.
Para comenzar el intercambio de llaves se acuerda el valor de *g* y *p*, estos valores son públicos, las funciones quedarían de la siguiente forma.

$$A \equiv g^a \pmod{p} \text{ y } B \equiv g^b \pmod{p}$$

A debe seleccionar un numero entero *a* que no debe revelar al igual *B* debe seleccionar un entero *b*. Y se calcula cada una de las expresiones.

A envía su resultado a *B* y *B* a *A*, debido al problema del algoritmo discreto es no es fácil definir el valor de *a* y *b* seleccionados.

Ahora el resultado de cada uno se multiplica por el número entero seleccionado y se obtiene el módulo de cada uno.

$$A' \equiv B^a \pmod{p} \quad \text{y} \quad B' \equiv A^b \pmod{p}$$

El valor de calcular A' y B' respectivamente es el mismo resultado.

$$A' \equiv B^a \equiv (g^b)^a \equiv g^{ab} \equiv (g^a)^b \equiv A^b \equiv B' \pmod{p}$$

Este valor es conocido como la llave intercambiada.
La figura 1.7 explica el intercambio de llaves.

Creación de parámetros públicos
Una entidad de confianza selecciona y publica un entero grande primo *p* y un entero *g*, teniendo un entero grande primo de orden F_P^*.

Cálculos privados	
Alice	**Bob**
Seleccionar un enero secreto *a*.	Seleccionar un entero *b*.
Calcular $A \equiv g^a \pmod{p}$.	Calcular $B \equiv g^b \pmod{p}$.

Intercambio de valores	
Alice envía *A* a Bob →	A
B	← Bob envía *B* a Alice

Cálculos privados futuros	
Alice	**Bob**
Calcula el número $B^a \pmod{p}$.	Calcula el número $A^b \pmod{p}$.
El valor secreto a compartir es $B^a \equiv (g^b)^a \equiv g^{ab} \equiv (g^a)^b \equiv A^b \pmod{p}$.	

Figura 1.7: Intercambio de llave Diffie - Hellman

1.9.3 RSA

La creación del algoritmo se debe a Ron Rivest, Adi Shamir y Leonard Adleman de ahí el nombre de RSA.
RSA es el algoritmo de llave pública más usado en todo el mundo, puede ser usado para el cifrado de información así como para la firma de documentos digitales. Este algoritmo se basa en la dificultad de factorización de grandes números enteros en conjunto con el problema del logaritmo discreto.

Factorización de grandes enteros

La base radica en la ecuación de tipo $N = pq$ donde *p* y *q* son dos números primos que multiplicados dan como resultado *N*.

Cuando estos números son muy grandes no existe un algoritmo eficiente capaz de realizar la factorización en tiempos razonables.

Existen diversos algoritmos para factorización de números, como es el caso del algoritmo de Pollard's el cual es rápido con números pequeños pero lento con números grandes.

La factorización más poderosa y simple que se puede utilizar es utilizando una identidad denominada diferencias de cuadrados $X^2 - Y^2 = (X + Y)(X - Y)$, que al igualar con *N* se obtiene $N + b^2 = a^2$.

La eficiencia depende de evaluar esta ecuación con la correcta selección de *b*, para encontrar el valor del cuadrado perfecto.

Algoritmo RSA

La base de este algoritmo es la generación de dos numero primos de gran tamaño *p* y *q* los cuales deben permanecer en secreto. Al producto de *p* y *q* se le conoce como *N*.

Además se debe seleccionar un numero *e* denominado exponente el cual debe cumplir con $\gcd(e,(p-1)(q-1)) = 1$.
De la generación de estos números se publica *N* y *e* los cuales son conocidos como la llave pública.

Para obtener el valor de *p* y *q* a partir de *N* y *e* es necesario recurrir a la factorización y dar solución al problema del logaritmo discreto.

Para el cifrado de información el mensaje en claro *m* debe estar codificado por un entero el cual debe ser mayor a uno y menor que *N*. Y se obtiene calculando la congruencia $c \equiv m^e \pmod N$, *c* es el mensaje cifrado a enviar.

Para realizar el descifrado de la información se debe calcular la congruencia $ed \equiv 1(\mathrm{mod}(p-1)(q-1))$, al ser el emisor el que genero *p* y *q* se conoce el valor de $(p-1)(q-1)$ por lo que se puede obtener *d* y se utiliza para realizar el cálculo de la congruencia $m' \equiv c^d \pmod N$. *m'* debe ser igual a *m* el mensaje descifrado.
En la figura 1.8 se muestra como funciona el algoritmo de RSA.

Bob	Alice
Creación de llave	
Selección de números primos *p* y *q*.	
Selección de un exponente de cifrado *e* con $\gcd(e,(p-1)(q-1))=1$.	
Publicar $N = pq$ y e.	
Cifrado	
	Seleccionar un texto claro a cifrar *m*.
	Usar la llave publica de Bob (N,e)
	para calcular $c \equiv m^e \pmod N$
	Enviar el texto cifrado *c* a Bob.
Descifrado	
Calcular *d* satisfaciendo	
$ed \equiv 1(\mathrm{mod}(p-1)(q-1)).$	
Calcular $m' \equiv c^d \pmod N$	
Entonces *m'* es igual al texto claro *m*.	

Figura 1.8: Algoritmo RSA creación de llaves, cifrado y descifrado.

1.10 Algoritmos criptográficos para integridad de la información

Los algoritmos criptográficos de funciones hash son muy versátiles ya que su uso abarca desde la autenticación de mensajes binarios hasta la firma de documentos digitales.

Autenticación de mensajes

Si es requerido verificar la integridad de un mensaje es necesario validar que la información recibida es exactamente igual que la información transmitida, con el uso de funciones hash es posible la autenticación de mensajes.

La implementación de mecanismos hash a partir de una mecanismo de llave simétrica y un bloque de información se le conoce como Código de Autenticación de Mensajes (MAC) o también denominadas funciones hash con clave.

Firma digital

Una aplicación importante para la función hash es la firma digital, en la cual el hash se cifra con una llave privada y si requerido verificar la integridad de la información se utiliza la llave pública.

En este caso la modificación de un solo bit del mensaje provocaría la no concordancia de la firma digital. La figura 1.9 muestra los insumos para la firma digital.

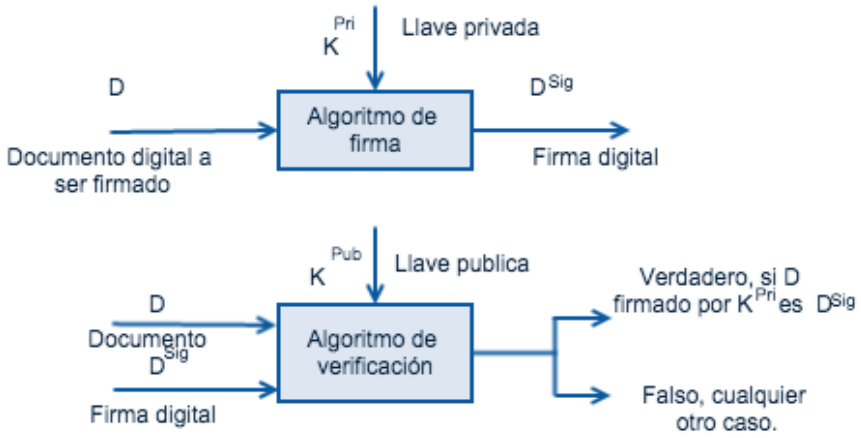


Figura 1.9: Componentes de la firma digital

1.10.1 Algoritmo de Hash Seguro (SHA)

El Algoritmo de Hash Seguro actualmente se encuentra en su versión número dos la cual se conoce como SHA-2 y abarca hash de longitud 256, 384 y 512.

El algoritmo SHA-512 toma un mensaje de entrada con una longitud máxima de 2^{128} bits y produce una salida de 512 bits que representa el hash del mensaje. La entrada es procesada en bloques de 1024 bits.

El proceso para la obtención del hash en SHA-2 se realiza en cinco pasos.

Añadir relleno de bits: Al mensaje es rellenado hasta que su longitud sea congruente con 896 modulo 1024, el relleno siempre se agrega aun cuando se tenga la longitud deseada.

Añadir longitud: Un bloque de 128 bits es agregado al mensaje y contiene la longitud del mensaje antes del relleno de bits.

Inicialización del búfer hash: Un búfer de 512 bits es usado para intermediar al inicio y al final de la función hash, es representado por ocho registros (a, b, c, d, e, f, g, h) de 64 bits.

Procesamiento de mensajes: Cada bloque de 1024 bits es procesado en 80 rondas. Cada ronda toma los valores del búfer y los actualiza modificando con cada iteración los valores del búfer.

Salida: Después de procesar todos los bloques de 1024 bits se suman los valores del búfer inicial con los valores del búfer final. La representación del resultado se realiza concatenando los valores del búfer.

Actualmente se encuentra en etapa de estandarización el algoritmo de hash para SHA-3, el cual fue seleccionado por el NIST en el 2012. El algoritmo ganador se denomina Keccak y pertenece a la familia de las funciones esponja.

Estas funciones toman un flujo de entrada de cualquier longitud y producen un flujo de salida de una longitud deseada. Figura 1.10 explica gráficamente el algoritmo SHA.

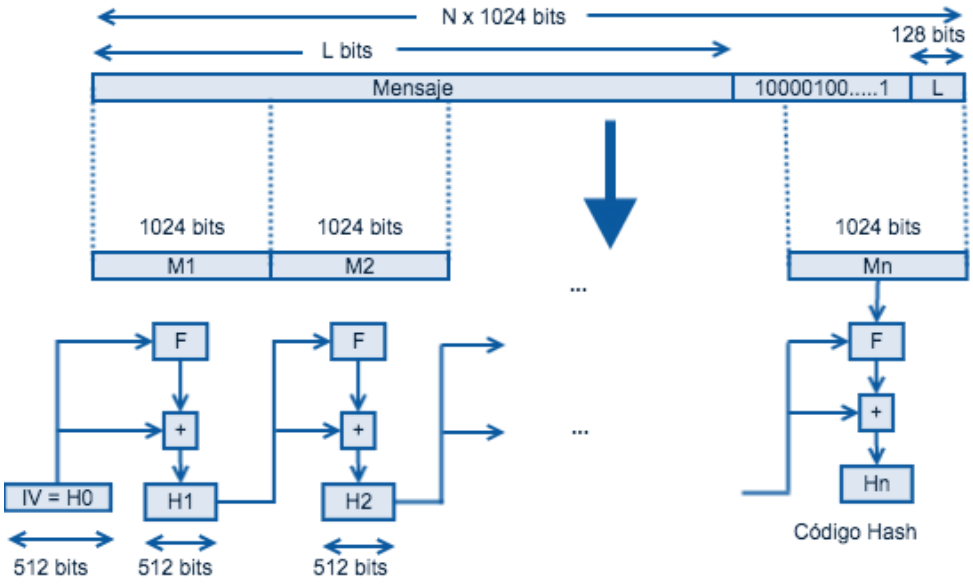


Figura 1.10: Generación de hash usando SHA-512

1.11 Certificados digitales

Los certificados digitales siguen el estándar X.509 que está basado en el uso de criptografía de llave pública y firma digital.

La expedición de certificados digitales lo realiza una autoridad certificadora (CA) la cual avala la autenticidad del certificado, por tal motivo la CA firma el contenido del

certificado digital y expone su llave pública para la validación del certificado. Figura 1.11, firma y validación de un certificado digital.

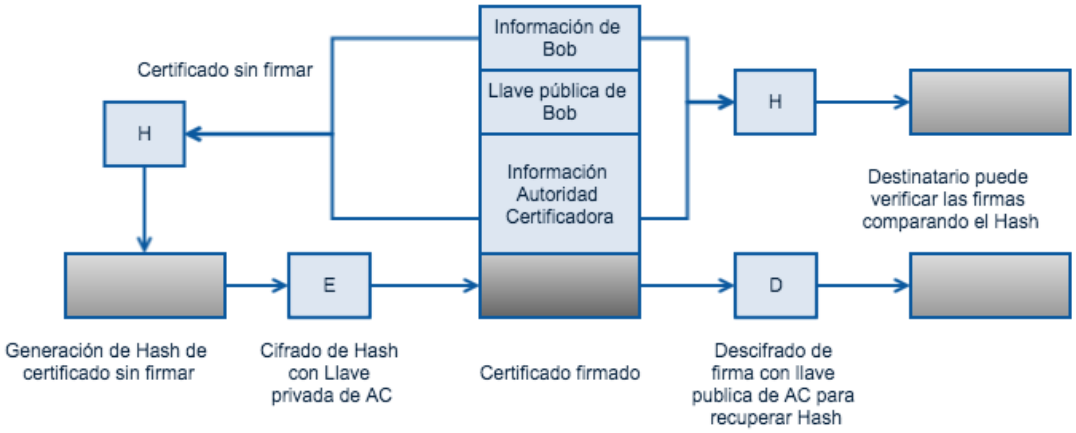


Figura 1.11: Creación de un certificado digital.

El estándar X.509 define un esquema de datos para almacenamiento de información en el certificado. Figura 1.12 muestra las versiones del estándar X.509.

- Versión: Actualmente existe hasta la versión tres, cada versión tiene correcciones y se agregaron o suprimieron campos del esquema.
- Serial: Es un numero único asociado a la autoridad certificadora.
- Identificador del algoritmo de firma: Es el algoritmo usado para firmar el certificado digital.
- Nombre del emisor: Nombre de la autoridad certificadora.
- Periodo de valides: Fecha de inicio y fecha final de validez del certificado.
- Nombre del sujeto: Nombre del sujeto al que se refiere el certificado digital
- Información de la llave publica del sujeto: Identificador del algoritmo con que fue generada la llave pública y la llave pública.
- Extensión: Conjunto de campos para agregar información.
- Firma: Es el hash que incluye los campos del certificado digital, la cual es generada con la llave privada de la CA.

El esquema de datos donde se almacena la información del certificado se basa en el estándar X.500 el cual asocia la información a nivel de aplicación y se maneja como un conjunto de objetos asociados a atributos organizados de una manera lógica y jerárquica.

Cada atributo lleva asociado un identificador único definido por una letra o conjunto de letras. En el caso de las extensiones se pueden definir esquemas con base a X.500 para agregar nuevos atributos.

La versión dos de X.509 contenía restricciones las cuales se agregaron como extensiones opcionales, y consisten en un identificador de extensión, un indicador de criticidad y el valor de la extensión. El indicador de criticidad indica si el valor puede ser ignorado.

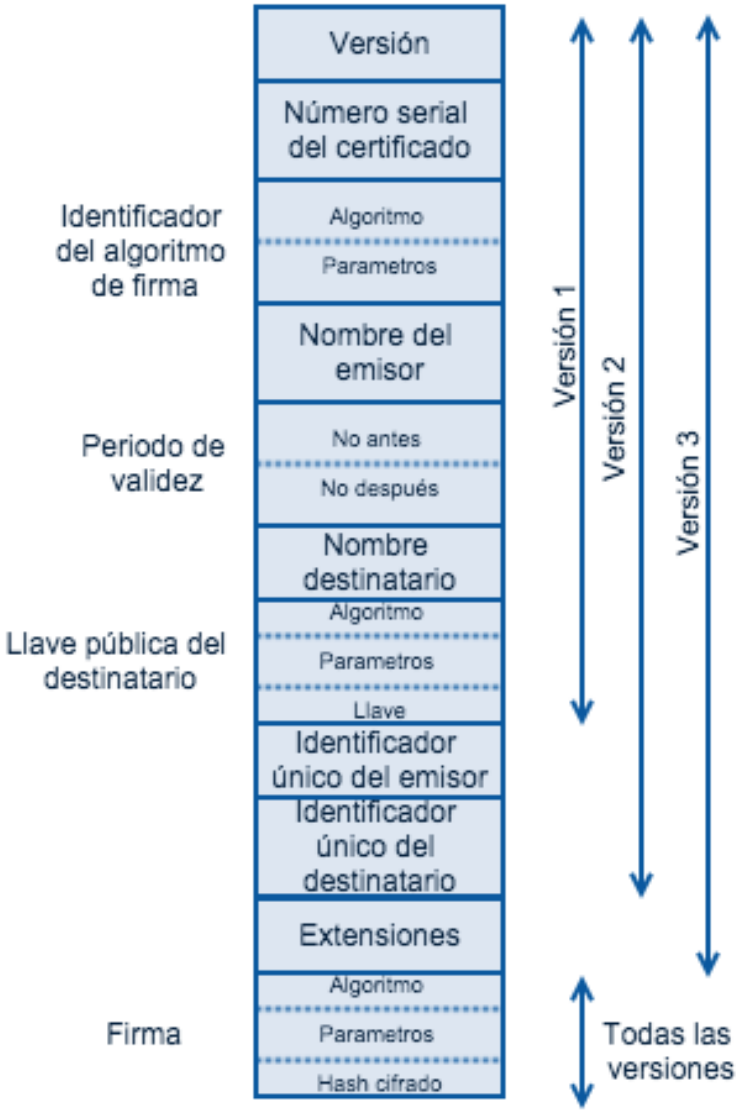


Figura 1.12: Formato X.509

Capítulo 2

Desarrollo de software

2.1 Lenguajes de programación

La función principal de un lenguaje de programación es indicar a la computadora una serie de operaciones que se realizaran en un orden específico. Dichos lenguajes siguen una cierta lógica y sintaxis que varía de un lenguaje a otro.

Los lenguajes de programación se dividen en compilados e interpretados.

Los lenguajes compilados requieren un procesamiento mediante un compilador el cual traduce de un lenguaje de programación y genera un programa equivalente pero de un nivel de comunicación intermedio o de bajo nivel.

En un lenguaje interpretado (script) la validación de sintaxis y la evaluación de instrucciones se realizan en tiempo real por un intérprete que ejecuta las instrucciones correspondientes.

Algunos de los beneficios de contar con un lenguaje compilado es tener comunicación nativa con el sistema operativo o directamente con la computadora por lo que se refleja en velocidad al procesar las operaciones. Mientras que los lenguajes interpretados es más simple su modificación e implantación ya que no requieren compilación.

Programación Orientada a Objetos (POO)

En los lenguajes de programación modernos se pueden definir atributos y encapsularlos para formar un objeto. También se encuentran mecanismos para la interacción entre objetos llamados métodos. Para agrupar los atributos y los métodos de un objeto se encapsulan en una clase. Una clase puede hacer referencia a varios objetos y también puede heredar las características de un objeto.

2.2 API

Si un conjunto de clases contienen funcionalidad en común para dar solución a un problema, se pueden agrupar para formar una API (Application Programming Interface) así esta funcionalidad puede ser reutilizada por otra aplicación dejando a un lado el cómo se realiza y enfocándose solamente en los valores de entrada y salida de las clases de la API.

2.3 Patrones de diseño de software

Al implementar funcionalidad en una aplicación se presentan problemas que de acuerdo a sus características pueden ser resueltos con base a un patrón de diseño de software. Los patrones de diseño de software brindan una solución probada y documentada a problemas con características similares. Algunos de los patrones se utilizan para la inicialización y configuración de objetos (Patrones creacionales), otros para separar funcionalidad (Patrones estructurales) así como de comunicación entre clases (Patrones de comportamiento).

Patrón Modelo – Vista – Controlador (MVC)

El patrón MVC separa las operaciones de persistencia de datos, la presentación y las acciones del usuario en tres clases. Figura 2.1 muestra el diagrama de comunicación.

Modelo: Administra el comportamiento de la información en la aplicación. Se utiliza como una capa de acceso a datos.
Vista: Administra la información a desplegar.
Controlador: Se encarga interpretar las acciones del usuario notificando a la vista o al modelo de los cambios ocurridos.

Una ventaja de la separación de operaciones es la de poder validar cada módulo por separado. Además la funcionalidad de la aplicación no está asociada con la vista, por tal motivo la actualización visual no afecta la funcionalidad de la aplicación.

2.4 Estándares de comunicación

Actualmente el protocolo de comunicación más utilizado en los navegadores web es Hypertext Transfer Protocol (HTTP) y funciona mediante intercambio de mensajes tipo solicitud/respuesta a través de una conexión sin estado es decir no guarda información sobre conexiones anteriores.

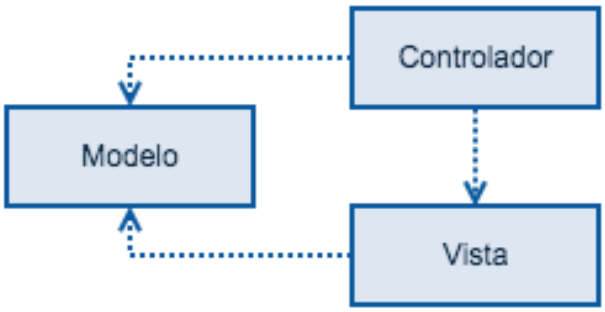


Figura 2.1: Estructura de clases MVC

Cada solicitud comienza con un método de petición que identifica el propósito de la solicitud y anticipa el tipo de respuesta del servidor. Dichos métodos se han utilizado para definir una arquitectura de reutilización en sistemas basados en red denominada REpresentational State Transfer (REST).

Los métodos utilizados por REST son POST, GET, PATCH y DELETE los cuales son asociados con operaciones a base de datos como lectura, escritura, actualización y borrado.

Estos métodos son implementados en el servidor para ser consumidos por un cliente. Una forma de consumir estos métodos es mediante peticiones HTTP asíncronas que permiten recuperar información del servidor sin recargar la página.

Esto se realiza con peticiones en segundo plano desde Javascript y actualizando el Document Object Model (DOM) de la página, es decir, los datos dinámicos de la página se cargan desde el lenguaje de script y son agregados como objetos al modelo del documento. Esta técnica es conocida como Asynchronous JavaScript And XML (Ajax[1]).

2.5 Framework

Para realizar una aplicación de una forma ágil y confiable se hace uso de los frameworks de programación. Los frameworks otorgan un conjunto estandarizado de conceptos, prácticas y criterios que se enfocan en una problemática en particular. Los framework hacen uso de las APIS para brindar funcionalidad, también utilizan

[1] No necesariamente la información se transmite como XML también se utiliza JSON.
AJAX, ultima modificación diciembre 2014 , http://es.wikipedia.org/wiki/AJAX

patrones de diseño para extender funcionalidad o solucionar problemas, que al utilizar el framework ya no es evidente y se considera transparente para el desarrollador.

Existen frameworks tanto para el desarrollo de aplicaciones cliente que se ejecutan a través de un navegador web ya sea para administrar conexiones o realizar interfaces graficas amigables, y de una manera similar se utilizan frameworks del lado del servidor los cuales ayudan con la gestión de operaciones con una base de datos o para gestión de las vistas.

2.5.1 Frameworks servidor

Spring Framework

Es un conjunto de herramientas de software que al ser integrado en una aplicación ofrece implementaciones a problemas de código, dejando de lado el desarrollo y configuración de código de funcionalidades genéricas para centrarse en aspectos de negocio. Figura 2.2. muestra los módulos más utilizados del framework

Unos de los patrones más utilizados por Spring es el de Dependency Inyection (DI) con el cual los objetos son inicializados a partir de su contenedor, este contenedor se encuentra definido en un XML por la simpleza de poder modificarlo sin la necesidad de recompilar la aplicación. Ya que el objeto se encuentra definido en XML y cargado en el contenedor Spring puede inyectar la dependencia en un objeto mediante anotaciones[2].

Spring Data interactúa directamente con el manejador de base de datos a través de objetos de programación mediante interfaces (Repositories) que heredan las operaciones de búsqueda, guardado y actualizado de base de datos.

En ocasiones las operaciones de base de datos se realizan en conjunto, y es necesario garantizar la correcta ejecución del bloque de operaciones esto se realiza con la anotación @Transactional(readonly = false) definida en el inicio del método para acceso a datos.
Spring Web contiene un módulo para el manejo de peticiones así como de respuestas, la anotación @RequestMapping permite definir la ruta de acceso de cada petición así como el método de acceso ya sea GET, POST, etc. Las respuestas se envían dentro del cuerpo del mensaje con un formato JSON[3] utilizando la anotación @ResponseBody.

---

[2] Las anotaciones en Java son elementos para almacenar información que puede ser accedida en tiempo de ejecución y no afectan directamente su funcionalidad.
Annotations, consulta febrero 2015 ,
http://docs.oracle.com/javase/1.5.0/docs/guide/language/annotations.html

[3] Es un formato de intercambio de información basado en notación de objetos de JavaScript, fácil de leer y escribir tanto para humanos como para maquinas.

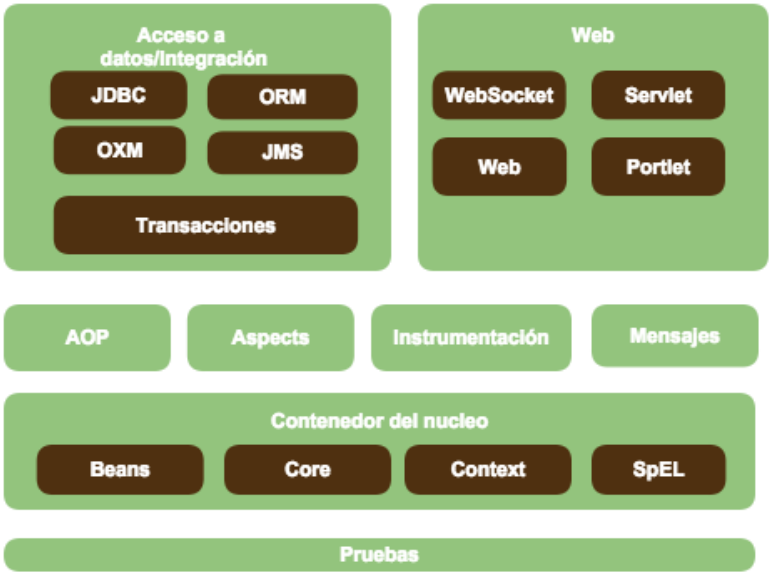


Figura 2.2: Estructura de Spring Framework

El acceso a objetos del contenedor desde una clase de servicio se realiza mediante la anotación @Autowired, su función es inyectar la implementación almacenada en el contenedor.

Tiles

Es un framework que hace uso del patrón composite para la generación de interfaces web. Esto se logra definiendo secciones de página que se pueden reutilizar, con ayuda de tiles estas secciones se unen para formar una página web completa.
La definición de las secciones web se realiza en un archivo XML las cuales se almacenan en el contenedor de Spring para que sean accesibles desde los objetos de vista al momento de acceder a una página web.

2.5.2 Frameworks cliente

En una aplicación web se definen elementos de interacción con el usuario, estos eventos se envían al servidor para brindar un comportamiento específico en la aplicación o para obtener información que desplegara el cliente.
Al realizar la implementación de control en el cliente hay elementos de código que son recurrentes y que realizan la misma funcionalidad, esto se puede aislar en código de JavaScript[4] reutilizándolo en múltiples páginas.

JSON, consulta febrero 2015, http://www.json.org/

[4] Es un lenguaje de programación ejecutado en el cliente vía navegador web, es interpretado basado en el estándar ECMAScript y orientado a objetos.

Entonces la implementación del lado del cliente se debe ordenar para tener un pleno control en caso de quitar o agregar funcionalidad, aislando el control de la vista y accediendo a los datos desde el control o la vista.

AngularJS

Angular permite segmentar el desarrollo en el cliente otorgando flexibilidad para la implementación del control, la vista y el acceso a datos. La comunicación entre la vista y el control se lleva a cabo mediante directivas que son interpretadas por el control de Angular, permitiendo la interacción con variables y funciones desde el documento HTML.

Para la interacción entre los objetos del documento HTML y las directivas de Angular se utiliza JQuery[5] para interactuar con los elementos del DOM.

2.6 Java Applet

Las aplicaciones web cliente solo viven dentro del contenedor del documento HTML que se despliega en el navegador web y su visión está acotada a los elementos de integración que tenga el navegador web con la computadora.

Si es necesario interactuar con dispositivos de la computadora cliente como es el caso de una cámara, el micrófono o procesar un archivo binario es necesario utilizar un programa que realice la función de puente entre JavaScript y la computadora.

Un Applet es un programa que se ejecuta en una máquina virtual dentro de la computadora cliente y tiene interacción con el documento HTML. Al ejecutarse en una máquina virtual el acceso a los dispositivos de hardware no se realiza de forma directa por lo que son necesarias librerías nativas desarrolladas de acuerdo al sistema operativo.

Cabe destacar que los avances en el desarrollo de HTML5[6] y JavaScript han hecho posible el procesamiento de archivos de forma local pero la interacción con elementos nativos de la computadora aún se realiza con aplicaciones cliente asociadas al sistema operativo.

JavaScript, Mc Graw Hill Fourth Edition , John Pollock

[5] Librería en Javascript que permite manipular los elementos del documento HTML de una manera rapida y simple.
Jquery, consulta febrero 2015, http://jquery.com/

[6] Es la quinta revisión de HTML, se agregaron etiquetas para realizar operaciones de audio, video y datos, así como optimizaciones en el DOM.
W3 – HTML5, consulta febrero 2015, http://www.w3.org/TR/html5/

Capítulo 3

Sistemas operativos

3.1 Definición

Para comprender un Sistema Operativo (SO) primero se debe entender que elementos lo componen y la integración en el sistema. Una computadora es un conjunto de elementos de hardware interconectados entre sí, con el fin de realizar una tarea específica. Estos componentes de hardware son controlados por una Unidad Central de Procesamiento (CPU) que ejecuta comandos de bajo nivel para establecer una comunicación con los dispositivos. El problema radica en la ejecución de los comandos en un momento determinado sin interferir con los demás dispositivos conectados o que se agregaran, aunado a las tareas de usuario o hasta para enviar información por internet. Todas estas acciones las gestiona un sistema operativo en dos capas principalmente, tareas de sistema y tareas de usuario. La figura 3.1 muestra los componentes básicos en un sistema operativo.

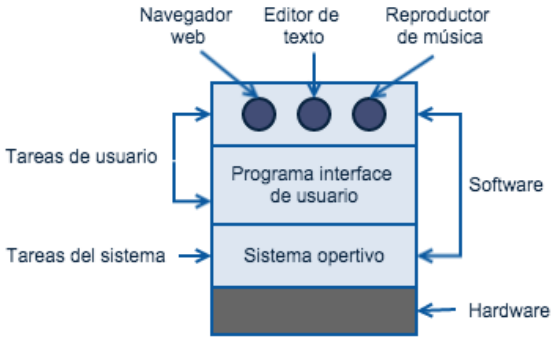


Figura 3.1: Estructura general de un SO

Las tareas de sistema se encargan de gestionar y establecer comunicación de bajo nivel con los dispositivos y las tareas de usuario se encargan de interactuar con los

comandos que ejecute el usuario para brindarle funcionalidad de una forma más transparente y simple. Algunas tareas de usuario se comunican con interfaces de las tareas del sistema para obtener o enviar información de bajo nivel.

3.2 Dispositivos de almacenamiento

Una de las tareas del sistema operativo es gestionar los dispositivos de almacenamiento como son los discos duros o las unidades de almacenamiento externo. Ya sea al momento de iniciar el SO o cuando sea detectado el dispositivo debe estar disponible para ser usado por las tareas de usuario.

3.3 Sistema de archivos

Para que los dispositivos de almacenaje puedan guardar información para posteriormente accederla los sistemas operativos gestionan los archivos otorgando características como son nombre, estructura, tipo, modo de acceso y atributos.
Cada sistema operativo tiene un sistema de archivos diferente, en el caso de Linux su sistema de archivos puede ser EXT3 o EXT4, NTFS para Windows y HFS para OS X.

Actualmente los dispositivos de almacenamiento externo USB optan por un sistema de archivos FAT32, aunque se considera antiguo la mayoría de los dispositivos como cámaras, celulares y televisiones lo siguen utilizando. Además es el sistema de archivos con soporte para lectura y escritura en todos los sistema operativos.

3.3.1 Directorios

Los directorios o carpetas sirven para almacenar la información en una computadora. Principalmente existen dos tipos de sistemas de directorios.
Un solo nivel o directorio raíz, en un solo directorio se almacenan todos los archivos es útil cuando se quiere localizar un archivo es utilizado por teléfonos o cámaras digitales.
Jerárquico, es el más utilizado ya que puede agrupar directorios y formar subdirectorios por lo que provee la capacidad de organizar no solo archivos sino también directorios. La figura 3.2 ejemplifica la estructura de directorios jerárquicos.

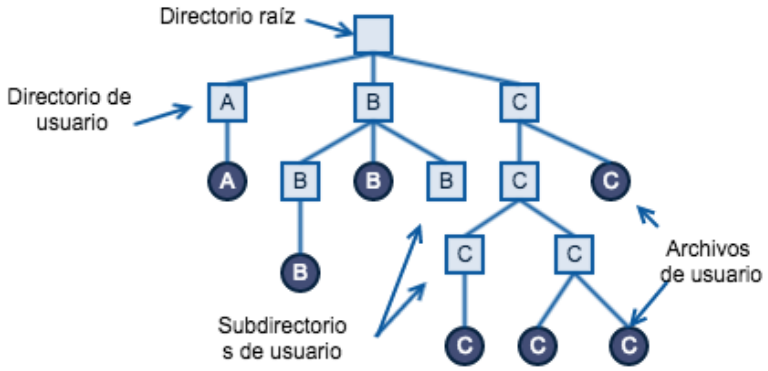


Figura 3.2: Sistema de directorios jerárquico.

En los sistemas operativos actuales el acceso a los directorios se realiza definiendo rutas de manera absoluta (ruta completa) o relativa (ruta relativa al punto actual), esto depende de la localización del archivo o directorio.
Para el caso de Windows el separador de carpeta utilizado es la barra diagonal inversa (\) mientras que en UNIX utiliza la barra diagonal (/).

3.4 Detección de dispositivos

En cada sistema operativo la detección de dispositivos de almacenamiento se realiza de forma diferente ocasionando que el mapeo del dispositivo con una unidad, un volumen o un punto de montaje no sea el mismo entre los sistemas operativos.

La detección de los dispositivos USB de almacenamiento implica una serie de interfaces con el sistema operativo para la extracción de información del dispositivo e identificar que drivers servirán para establecer la comunicación entre el hardware conectado y la computadora.

3.4.1 Detección de dispositivos en Linux

Es un sistema operativo de código libre basado en UNIX, actualmente corre en diversas arquitecturas de procesadores. El código es mantenido por desarrolladores de todo el mundo. Linux se utiliza en conjunto con las aplicaciones GNU también de código libre. Al núcleo de Linux en conjunto con GNU[7] se le denomina proyecto GNU/Linux o distribución GNU/Linux. Las distribuciones comparten el núcleo Linux pero agregan aplicaciones para generar personalizaciones y obtener un sistema operativo a medida. La mayoría de las aplicaciones son del proyecto GNU pero también pueden incluir aplicaciones propietarias.

Gestión de dispositivos en Linux

Udev se encarga de administrar los dispositivos al ser detectados por el núcleo con ayuda del demonio[8] udevd, cuando recibe un evento de conexión o desconexión de un

[7] GNU es un acrónimo recursivo de ¡GNU No es Unix!. Proyecto iniciado por Richard Stallman con el fin de tener aplicaciones de software libre bajo la licencia GPL para su utilización, modificación y redistribución.
GNU, consulta febrero 2015, http://www.gnu.org/

[8] Demonio, nomenclatura utilizada en los sistemas *NIX referente al proceso que se ejecuta en segundo plano.

dispositivo se extrae la información de los controladores necesarios para su funcionamiento mediante sysfs. Toda la información del dispositivo se encuentra dentro de su descriptor, los atributos utilizados como es bcdUSB indica las especificaciones para la versión de USB, el bcdDevice indica el número de revisión del dispositivo, en conjunto con el idVendor y el idProduct son utilizados para la selección del driver más óptimo para el dispositivo. Una vez identificado el dispositivo se valida su configuración así como las reglas (si es que tiene asociadas) para posteriormente cargar los controladores necesarios con la configuración y reglas definidas. Por último se genera un enlace simbólico dentro de /dev.

Hasta este momento el dispositivo ya se encuentra listo para el SO, pero si el dispositivo es de almacenamiento aún no cuenta con un punto de montaje.
El montaje se define en /etc/fstab donde se indica el enlace simbólico asociado en *dev* y un directorio del sistema de archivos del SO. El punto de montaje se define con un tipo partición así como los permisos de acceso. Otra forma de validar el punto de montaje es utilizando el comando mount donde se encuentra la asociación del enlace simbólico en /dev y el directorio de montaje.

En Linux los directorios comunes para montar los dispositivos de almacenamiento son /mnt y /media.

3.4.2 Detección de dispositivos en OS X

La familia de los SO UNIX comienza en el año de 1969 cuando Ken Thompson y Dennis Ritchie comenzaron a trabajar con una computadora PDP-7 en los laboratorios Bell. UNIX comenzó a tomar importancia cuando incluyo un ensamblador para la PDP-11/20, un sistema de archivos y un procesador de textos.
Fue hasta 1973 cuando el sistema operativo fue reescrito a un lenguaje de programación desarrollado por Ritchie denominado “C”, con esto haciéndolo portable y cambiando la historia de los sistemas operativos.

En el año de 1985 Apple comienza por desarrollar un sistema operativo basado en UNIX BSD (Berkeley Software Distribution) enfocado a la programación orientada a objetos denominado NeXTSTEP.

Es a partir del año 2000 cuando Apple libera las primeras versiones de OS X 10.0 basadas en UNIX, hasta la versión 10.10 Yosemite liberada en 2014.

Demonio(Informatica), ultima modificación octubre 2014,
http://es.wikipedia.org/wiki/Demonio_%28inform%C3%A1tica%29

Gestión de dispositivos en UNIX

La gestión de los dispositivos se realiza se de manera análoga que en Linux, mediante la gestión de nodos de dispositivos en el directorio /dev.

Los nombres de los dispositivos de almacenamiento varían en razón de la distribución BSD utilizada pero se pueden identificar por los prefijos *disk* o *da*.

Los directorios más comunes para montar los dispositivos de acceso son /mnt, /media y /Volumes.

3.4.3 Detección de dispositivos en Windows

Es un sistema operativo para computadoras x86 desarrollado en el año de 1980, al principio comenzó con el nombre de MS-DOS y contenía operaciones muy similares a UNIX mediante línea de comandos. Posteriormente se agregaron módulos gráficos para su gestión y de ahí el nombre de Windows, ya que las aplicaciones se podían acceder a partir de una Interfaz Gráfica de Usuario (GUI, Graphical User Interface). Actualmente Windows puede ser ejecutado en arquitecturas de 64 bits así como en ARM.

Gestión de dispositivos en Windows

Cuando el sistema operativo recibe una notificación de conexión de un dispositivo el Plug And Play Manager valida que recursos requiere en el sistema operativo y así poder asignarlos al dispositivo.

A partir del descriptor del dispositivo se busca una fuente de instalación del driver, sí encuentra múltiples drivers se selecciona el que mejor coincida con el dispositivo de acuerdo a sus características. Posteriormente se instala el driver correspondiente al dispositivo.

Dependiendo del tipo de dispositivo se ejecutaran tareas para configurarlo con el sistema operativo, para el caso de las unidades de almacenamiento externo se asocia una letra para dejar el dispositivo accesible. La letra a asociar será en razón de los dispositivos que se encuentren presentes al momento de conectar el dispositivo y se asigna en el mismo orden del alfabeto. Las letras por default son C: Sistema operativo, A: Floppy, D: Cd-Rom.

3.5 Libusb para acceso a dispositivos

Al contemplar el acceso a dispositivos desde un sistema operativo causa excepciones al momento de interactuar con los puntos de montaje. Una forma de acceso a los dispositivos USB es utilizar libusb.

Libusb es una librería desarrollada en C que brinda a las aplicaciones acceso a dispositivos USB en diversos sistemas operativos. Cuenta con una serie de rutinas para controlar la transferencia de información hacia y desde el dispositivo USB, el acceso al dispositivo se realiza de forma nativa ya que no se utilizan los drivers del sistema operativo. Es un proyecto de código abierto y cuenta con la licencia GNU Lesser General Public License versión 2.1 o posterior.

Al contemplar libusb solo se utilizaría para homologar la obtención del descriptor del dispositivo USB conectado, ya que la comunicación a nivel de lectura y escritura con el dispositivo depende de los drivers de cada sistema operativo.

Capítulo 4

Diseño del framework

4.1 Historia

La intención de desarrollar un framework para interactuar con llaves asimétricas nace de la necesidad de poder interactuar con certificados digitales en aplicaciones tipo cliente servidor.

A partir de esta necesidad se desarrollaron una serie de librerías para poder interactuar no solo con certificados digitales sino también con llaves privadas codificadas en distintos estándares.

Es aquí donde comienzan la diversificación de las librerías por tratar de abarcar no solo aspectos de seguridad sino también de comunicación, almacenaje de llaves, firma de documentos y uso de llaves asimétricas para autenticación en aplicaciones.

Al no existir una solución integral que diera una solución es estos aspectos se comenzó consolidar una base de conocimientos enfocados en integrar criptografía dentro de aplicaciones existentes, es aquí donde el framework comienza a tomar forma.

4.2 Estructura

El framework esta formado principalmente por tres módulos

- Módulo de seguridad
- Módulo de dispositivo
- Módulo de mapeo

La figura 4.1 muestra las implementaciones básicas de cada módulo.

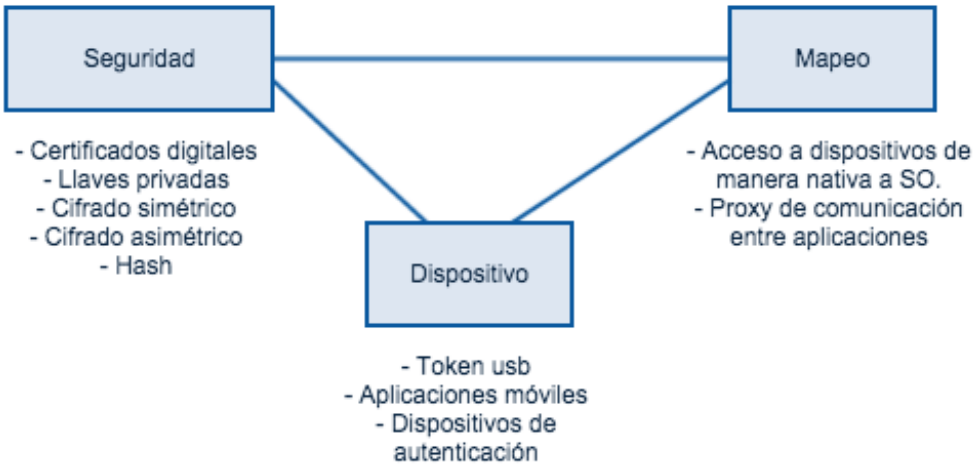


Figura 4.1: Implementaciones básicas por cada módulo

Cada módulo se encuentra diseñado para poder ser integrado de manera independiente en caso de solo necesitar funcionalidad especifica.

Un ejemplo sería el poder integrar a una aplicación un cifrado simétrico de registros en una base de datos, para este caso solo se utilizaría el módulo de seguridad de forma independiente.

Siguiendo con el mismo ejemplo, si se requiere utilizar una llave privada, el mismo módulo de seguridad provee la implementación para interpretar la llave privada. Pero modifiquemos un poco el panorama, en este caso la llave se encuentra almacenada en un dispositivo externo, para este caso se requeriría el módulo de mapeo para poder acceder al dispositivo de manera nativa desde nuestra aplicación.

El módulo de mapeo solo tiene la capacidad de poder encontrar el acceso a los dispositivos de manera nativa, sin importar el sistema operativo, la pregunta seria,

¿Como se pueden traducir las operaciones que puede realizar cada dispositivo?
Con el módulo de dispositivo, cada implementación traduce el alcance que puede tener cada dispositivo, ya sea solo de almacenaje de llaves o para autenticación.

Al unir estos módulos se pueden establecer diferentes combinaciones para diversos escenarios como la autenticación en sistemas operativos o hasta el uso de celulares para firma de documentos digitales.

A continuación se detalla el diseño de cada módulo así como la estructura de clases y métodos, esto con el fin de poder entender las capacidades actuales de cada módulo.

Posteriormente se proporciona un caso practico donde se unen los módulos en una arquitectura cliente-servidor web donde se ejemplifica la segmentación de los módulos y la integración con aplicaciones existentes.

Cabe destacar que dicha aplicaciones web también forma parte del framework ya que establece mecanismos de seguridad que pueden ser utilizados de manera independiente para aplicaciones web.

4.3 Estatus del framework

A continuación se muestra un diagrama con los elementos disponibles en el framework , en desarrollo y prospectos para implementación.

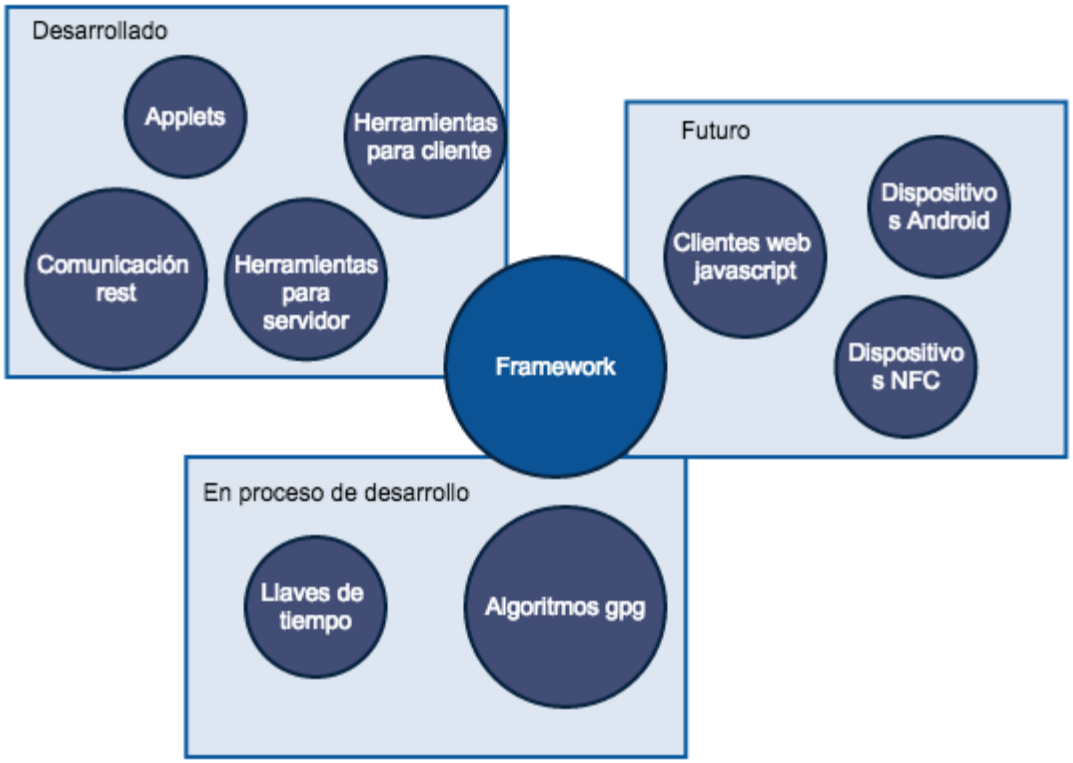


Figura 4.2: Estatus del framework

4.2 Módulo de seguridad

4.2.1 Definición del módulo

Para la interacción con elementos de seguridad el módulo cuenta con implementaciones de certificados digitales, llaves privadas cifradas, servicios de validación de certificados, contenedores de llaves y firma digital. Todo esto para interactuar en alto nivel dejando de lado las implementaciones de cada uno de los elementos de criptográficos.

El proveedor de seguridad se encarga de las implementaciones de los algoritmos a utilizar dentro del entorno de desarrollo. BouncyCastle[9] es el proveedor de seguridad utilizado como base para la implementación de los elementos de alto nivel.

Para la construcción del módulo se utilizan herramientas de gestión que ayudan no solo al momento de desarrollar código, sino también cuando se realizan pruebas al código, manejo de librerías o hasta para el empaquetado de las librerías.

La herramienta que ayuda a realiza estas acciones se llama Maven[10], utiliza un archivo POM (Project Object Model) donde se definen las dependencias, plugins a utilizar, reglas de compilación, pruebas de código así como estructura para empaquetar el proyecto, entre otras acciones.

La estructura del proyecto se encuentra definida de acuerdo a la figura 4.3

[9] BouncyCastle son una serie de librerías enfocadas al desarrollo de criptografía.
Bouncy Castle, consulta febrero 2015, https://www.bouncycastle.org/

[10] Maven es un proyecto de tipo de software libre con licencia Apache 2.0
Maven, consulta febrero 2015, http://maven.apache.org/

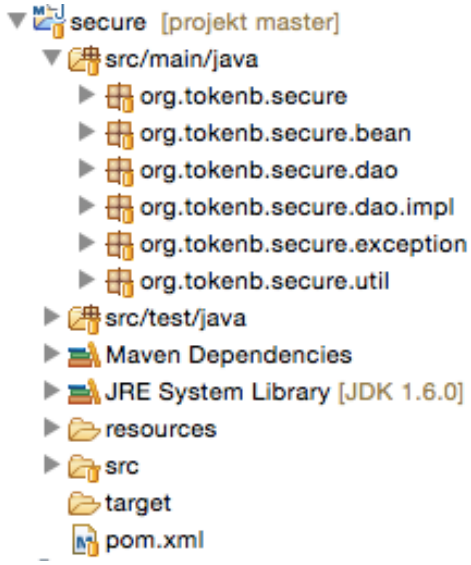


Figura 4.3: Estructura del módulo de seguridad

El código del módulo se encuentra dentro de la carpeta *src/main/java*. Las clases de acceso se encuentran definidas dentro del paquete *org.tokenb.secure.*

Las pruebas unitarias se encuentran definidas dentro de la carpeta *src/test/java* y se encuentran escritas con la libraría JUnit[11].

4.2.2 Generación de llaves seguras

La generación de llaves se realiza mediante la clase *SecureKeyPair*, genera en una sola clase la llave pública y la llave privada como un objeto de tipo *KeyPair*.

Se utilizan como argumentos la longitud de la llave (1024 o 2048 bits) y el algoritmo para utilizar las llaves(RSA o DSA). Esta clase se encuentra sobrecargada y se pueden definir combinaciones entre el algoritmo y la longitud de la llave.

La opción default para la generación de llaves seguras es RSA debido a sus llaves pueden ser utilizadas tanto para cifrar como para firmar, así como en una mayor longitud de la llave lo que se refleja en mayor seguridad.

La generación de los números pseudoaleatorios se realiza mediante la clase *SecureRandom*[12], que para ser instanciada se pasa como argumento *SHA1PRNG*. Esta

[11] JUnit es una biblioteca enfocada a la ejecución de pruebas unitarias definidas con código Java.
Junit, consulta febrero 2015, http://junit.org/

[12] Recomendaciones para generación de números pseudoaleatorios.
Randomness Recommendations for Security, publicación diciembre 1994
http://www.ietf.org/rfc/rfc1750.txt

clase se basa en un generador estadístico de números aleatorios en conjunto con un valor de inicio "*Seed*".
El *SecureRandom* puede ser sustituido con proveedores de seguridad como Bouncy Castle o IBM JCE los cuales contienen sus propios generadores de números pseudoaleatorios

4.2.3 ASN.1

Abstract Syntax Notation One es una notación para describir tipos de datos abstractos y sus valores. Existen tipos de datos universales para definir las clases de valores así como tipos privados que se definen de acuerdo a las características del dispositivo a utilizar. Estos tipos de campos siguen el estándar X.208 y X.500.

4.2.4 Codificación DER y PEM

Al momento de trabajar con certificados digitales y llaves privadas se pueden encontrar codificados principalmente en dos tipos de archivos DER y PEM.

DER

Distinguished Encoding Rules es la codificación de un certificado digital, una llave o información en general que se requiera definir en un formato portable, la notación utilizadas por DER es ASN.1. La información del esquema ASN.1 es almacenada en un archivo DER con formato en binario.

PEM

Privacy Enhanced Email es una codificación en base64[13] de un archivo DER.
Los archivos PEM al tener una codificación ASCII pueden ser enviados por texto, esta codificación es más amigable para la interacción entre personas.

Se encuentra definida la clase *Encoder* con métodos para codificar DER a PEM (*derToPem*), codificar PEM a DER (*pemToDer*) así como identificar objetos PEM dentro del archivo o flujo de datos (*isPEMObject*).

[13] Base64 es utilizada para almacenar o transmitir información que por razones de compatibilidad es utilizado ASCII.
The Base16, Base32, and Base64 Data Encodings, publicación octubre 2006, http://tools.ietf.org/html/rfc4648

4.2.5 Generación de certificados digitales

Los certificados digitales son archivos que contienen información para identificar una persona, una empresa o un sitio en internet mediante medios electrónicos.
Un certificado digital solo puede ser válido si un tercero lo avala, a este se le conoce como autoridad certificadora (CA).

Los certificados digitales contienen información acerca de los algoritmos utilizados para firma y validación, llaves públicas, información con detalles del sujeto asociado al certificado, firma y detalles de la autoridad certificadora que avala el certificado así como las fechas de generación y vencimiento.

El estándar utilizado para generar los certificados digitales es ASN.1 y generalmente se encuentra codificado con PEM para ser transmitido por correo electrónico. El estándar que siguen los certificados digitales es X.509[14], actualmente se encuentra en la versión 3 agregando campos con extensiones para agregar información al esquema de datos del certificado.

La generación de certificados digitales depende de una validación y autorización de una autoridad certificadora, con el fin de dar soporte legal ante controversias de autenticidad en firma de documentos digitales. La autoridad certificadora además de avalar la identidad de la persona o institución también lleva el control de los certificados vigentes así como de los revocados.

Dentro del módulo de seguridad se encuentra definida la clase *CertificateX509* utilizada interactuar con certificados digitales.

Para generar certificados digitales de una autoridad certificadora dentro de la clase se encuentran dos métodos, para codificaciones PEM *buildPEMCACertificate* o DER *buildDERCACertificate*.
Los campos a asociar al certificado digital de la autoridad certificadora se pueden ajustar para incluir más detalles que describan mejor la identidad y unicidad de la CA. Estos campos se definen de acuerdo a la notación Distinguished Name(DN) que asocia un nombre de catálogo con un valor (atributo=valor) cada DN se conectan por comas para describir un objeto. El formato utilizado suele ser UTF-8.
El número de años de vigencia se agrega como parámetro de los métodos.

Dentro de la clase *CertificateX509* también se encuentran los métodos para crear certificados digitales de clientes.

---

[14] Contiene especificaciones de formato y semántica del certificado, así como reglas de codificación.
Internet X.509 Public Key Infrastructure Certificate and CRL Profile, publicación enero 1999, https://www.ietf.org/rfc/rfc2459

Tanto los certificados cliente como los de CA se generan con la versión 3 del estándar X.509. La definición de los atributos DN del sujeto se encuentran definidas dentro de la clase *X509Principal* como elementos estáticos finales, cada objeto entrega un *DERObjectIdentifier* para ser agregado dentro de la definición del certificado.
Si es requerido agregar un identificador en particular se debe generar un nuevo objeto *DERObjectIdentifier*.

Los números de serie son los identificadores del certificado digital y van asociados con la autoridad certificadora que lo genero. Se pueden encontrar dos certificados digitales con el mismo número de serie pero nunca firmados por la misma autoridad certificadora.

Los números de serie se encuentran definidos en hexadecimal y son enteros positivos grandes.

Las fechas de vigencia del certificado digital son variables y pueden ser generadas con intervalos de horas o años. Por default la fecha final está definida en 4 años pero se puede modificar con el atributo *YEARS_VALID_CERT_CLIENT*.

El método para generar certificados digitales cliente admite como argumento un *KeyPair* para agregar solo la llave pública al certificado.

Por último se agrega el nombre del algoritmo con el que se firma el certificado digital, por default se tiene definido *SHA1WithRSAEncryption*.

Al construir un certificado digital es necesario seleccionar una codificación, por tal motivo se cuenta con dos métodos, *buildPEMClientCertificate* y *buildDERClientCertificate* para codificación PEM y DER respectivamente.

Para poder integrar la librería se cuenta con métodos que facilitan el manejo de certificados digitales independientemente donde y como se encuentren definidos.

Para leer los certificados digitales de un archivo en disco existe el método sobrecargado *readX509CertificateFrom*, como argumento se puede proporcionar la ruta completa del archivo o un objeto de tipo *File*.

Otros métodos importantes son *readX509CertificateUTF8String* el cual extrae a partir de una cadena de texto codificada en UTF-8 el certificado digital así como *readX509CertificateByteArray* a partir de un arreglo de bytes construye el certificado digital.

Directamente con el certificado se pueden validar las fecha inicial(a partir de cuándo es vigente) y final (hasta cuando es vigente) esto se realiza con el método *isDateValidCertificate*. Para validar si el certificado digital fue generado por una autoridad certificadora existe el método *isCertBuildInThisCA*.

4.2.6 Revocación de certificados

Cuando un certificado digital se ve comprometido o es necesario actualizar su contenido, antes de generar un nuevo certificado digital se requiere revocar el certificado para anular su autenticidad.

Un certificado digital por si solo tiene una fecha de vigencia pero solo hasta que se cumple deja de ser válido, el problema radica cuando es necesario anular el certificado antes de la fecha de vigencia.

Los certificados digitales son asíncronos para su funcionamiento, pero existen mecanismos para validar el estado de los certificados. Las Certificate Revocation List (CRL) ofrecen periódicamente un listado con los números de serie de los certificados que fueron revocados. Esto no ofrece una garantía que el certificado sea vigente ya se depende de la frecuencia en la que sea actualizada la lista además que el listado es incremental y las consultas se vuelven ineficientes.

Un mecanismo de validación en línea denominado Online Certificate Status Protocol (OCSP) permite realizar la consulta en tiempo real de un certificado en específico a diferencia de consultar toda una lista de revocación.

La notación utilizada por OCSP es ASN.1 y utiliza http como canal de comunicación para transferir la información. Cabe destacar que existen implementaciones que utilizan el canal de comunicación HTTPS o SSH.

La validación de certificados revocados se realiza con la clase *OCSP*, con el método *isValid* se realiza la consulta al servidor utilizando el número de serie del certificado a validar obteniendo como respuesta el estatus del certificado en forma de objeto que contiene los detalles del estatus.

4.2.7 Generación de llaves PKCS #8

Al momento de generar un juego de llaves únicas la llave publica queda a disposición del certificado digital y como su nombre lo indica puede ser dominio público por lo que no existen inconvenientes en estar expuesta. El problema radica en la otra llave, la privada, ya que dejarla expuesta como en el caso del certificado digital causaría un serio problema de seguridad en caso de caer en manos equivocadas. Es por esta razón que las llaves privadas son cifradas con un algoritmo simétrico y así prevenir el uso de la llave privada si esta se ve comprometida.

La generación de llaves privadas utiliza el mismo esquema de notación que los certificados digitales denominado ASN.1 al igual que las codificaciones utilizadas para almacenar la llave privada pueden ser DER y PEM.

Debido a que las llaves privadas se encuentran cifradas, para su generación e interpretación son construidas con base en el estándar PKCS #8, con el cual se describe la sintaxis para cifrado de llaves privadas, la definición de los algoritmos que pueden ser utilizados para cifrar una llave privada así como la interpretación de las llaves para ser descifradas.

Para el cifrado de las llaves privadas son utilizados algoritmos simétricos como el AES en sus versiones 128,192 y 256 bits. El algoritmo DES en su versión 3DES es uno de los más utilizados para el cifrado de llaves privadas.

Para garantizar una mayor seguridad en la llaves pasan por un proceso denominado Cipher-Block Chaining (CBC) que realiza una operación de XOR a cada bloque del texto en claro antes de ser cifrado, requiere de un vector de inicialización en el primer bloque para comenzar con el cifrado. La figura 4.4 muestra gráficamente como funciona CBC.

La definición de los algoritmos implementados con sus variantes se localiza en la clase *PKCS8Generator* y se encuentran definidos como atributos estáticos.

La clase *PKCS8Key* contiene las implementaciones para interactuar con llaves privadas cifradas. Contiene métodos para construir llaves privadas cifradas con AES256 utilizando el método *buildPEMKeyAES256CBC* o utilizando el algoritmo 3DES *buildPEMKeyDES3CBC*. Ambos métodos tienen sus variantes para construir llaves en codificación DER.

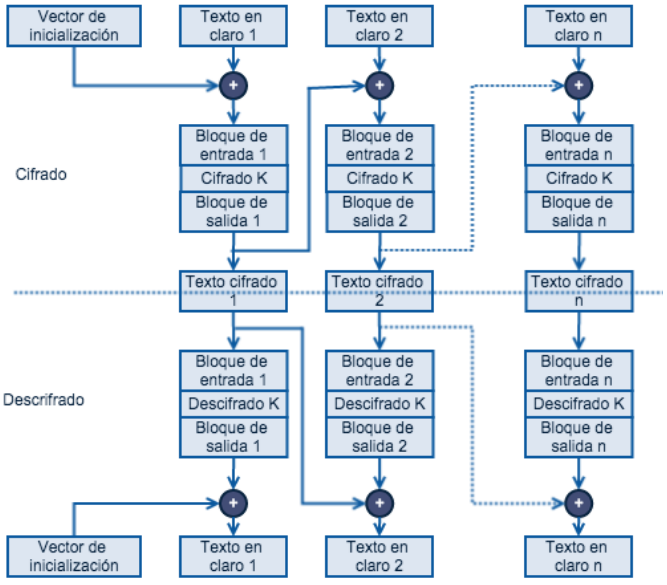


Figura 4.4: CBC Mode

La interpretación de las llaves privadas cifradas se realiza con los métodos *readEncryptedKeyFrom* para lectura de un archivo desde un dispositivo de

almacenamiento, *readEncryptedKeyFromByteArray* interpretación desde un flujo de datos y *readEncryptedKeyFromUTF8String* para interpretación desde una cadena de texto con codificación UTF8.

4.2.8 Autoridad de registro

Al tener las implementaciones de los certificados digitales así como de las llaves privadas cifradas es requerida la generación en conjunto asociando estos dos componentes con un juego de llaves único. Además de la generación de los juegos de llaves utilizados por la Autoridad Certificadora para firmar los certificados cliente.
En esta etapa también se valida la autenticidad de la información contenida en el certificado digital.

En la clase *RegistrationAuthority* se encuentran definidos los métodos *buildPKIElements* para generar el juego de llaves (certificado digital y llave privada cifrada) de la Autoridad Certificadora en codificación DER y el método *getPKIElements* para leer las llaves de la autoridad certificadora a partir de una ruta de un dispositivo de almacenamiento.

Para la generación de las llaves cliente se encuentra definido el método *buildPEMCertificate* que construye el certificado digital y la llave privada cifrada cliente con ayuda de la llave privada de la autoridad certificadora para firmar el certificado cliente. Las codificaciones implementadas son DER y PEM.

4.2.9 Almacenamiento de llaves

Una vez que el certificado digital es emitido por la autoridad certificadora se emiten dos archivos asociados únicamente por el juego de llaves pública y privada.
El almacenaje de estos archivos puede estar en dispositivos diferentes pero al realizar algunas operaciones es necesario trabajar con ellos en conjunto. Por tal motivo se utiliza el estándar PKCS #12 para almacenar conjuntamente la llave privada y el certificado digital en un solo archivo cifrado.

Dicho archivo puede almacenar diversos certificados digitales y llaves privadas asociándolas en un mismo contenedor cifrado además de estar firmado ante cualquier modificación. La asociación se realiza mediante nombres o alias. El archivo se puede encontrar definido con la extensión p12 o pfx.

La generación de los archivos p12 se realizan con la clase *PKCS12* específicamente con el método *pakageKeys* indicando el destino donde se genera el archivo p12, la llave privada y el certificado digital, así como un alias para vincular las llaves.

La lectura del archivo se realiza mediante el método *readKeys* indicando la ubicación del archivo p12, la contraseña y de que alias se quiere extraer las llaves.

También se puede realizar una consulta con el método *listAlias* para extraer el contenido del archivo p12 y conocer los alias disponibles.

4.2.10 Firma y validación de información

Una de las aplicaciones que se le dan al certificado digital y la llave privada es la de firmar documentos electrónicos. El algoritmo de firma electrónica garantiza la integridad del documento así como la identidad del firmante. Actualmente en México el Servicio de Administración Tributaria (SAT) es una de las de dependencias de gobierno que funge como Autoridad Certificadora y por ende utiliza el juego de llaves para diversos trámites electrónicos.

El algoritmo de firma electrónica se basa en la obtención del hash del documento a firmar, cabe señalar que la firma electrónica no modifica la estructura ni contenido del archivo y se genera como adicional al documento firmado. Ya que el hash del documento es único para cada archivo es utilizado para validar la integridad del archivo, con ayuda de la llave privada el hash es cifrado para la obtención de la firma electrónica. La figura 4.5 muestra el proceso de firma digital.

Debido a que la llave publica que se encuentra dentro del certificado digital se comparte para posteriormente poder realizar la validación.

Previo a la firma de todo documento digital es necesario realiza la validación del certificado, esto se debe realizar revisando los periodos de vigencia del certificado así como la revisión mediante listas de revocación o servicios OCSP.

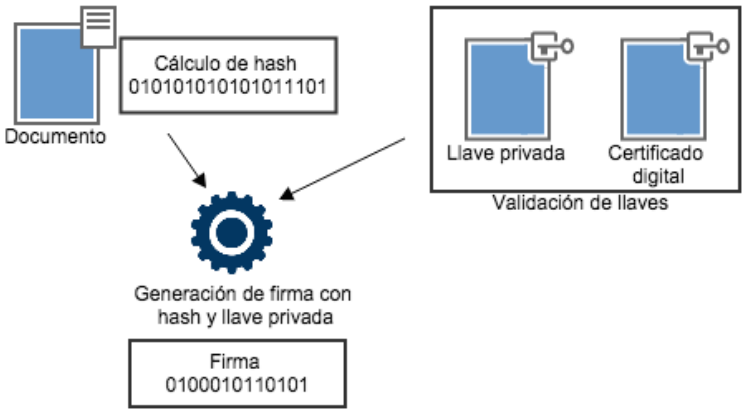


Figura 4.5: Firma

Para la validación de la firma, el mensaje es descifrado con la llave publica contenida en el certificado digital , el resultado obtenido es el hash del documento antes de la

firma, por lo que es necesario volver a obtener el hash y realizar su comparación. Sí los dos hash son iguales indica que el documento no ha tenido ninguna modificación y pertenece a al certificado digital. La figura 4.6 muestra el proceso de validación de firma digital.

Pero si la firma no puede ser descifrada para obtener el hash indicaría que no pertenece a dicha llave publica y no se encuentra asociado con el firmante. De manera similar si el hash cifrado no coincide con el hash de validación indicaría una modificación posterior a la firma del documento por lo que la firma no sería válida.

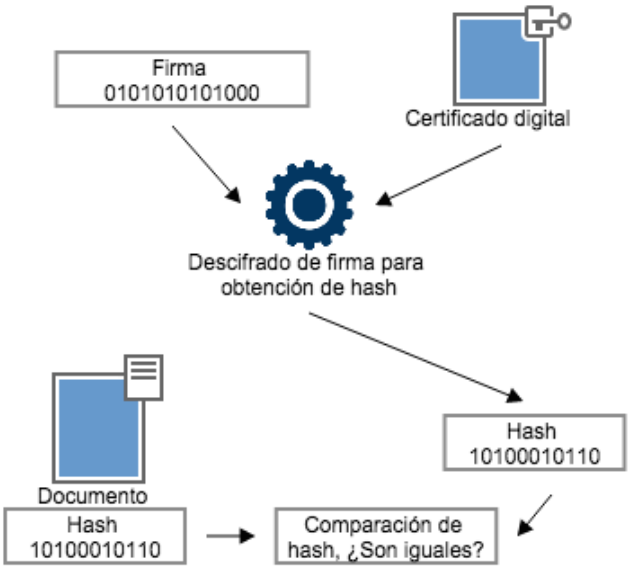


Figura 4.6: Validación de firma

El firmado y validación de documentos se realiza mediante la clase *SignatureData*. *Sign* es el método definido para firma de documentos, se encuentra sobrecargado para poder definir el algoritmo de firma.

La validación de la firma se realiza mediante el método *verify*, también se encuentra sobrecargado para poder definir el algoritmo de firma.

Los algoritmos definidos dentro de la clase son las variantes de SHA para 160, 256 y 512 bits utilizando los algoritmos de llaves asimétricas RSA y DSA[15].

[15] DSA es un algoritmo de firma digital, rápido para cifrar y más lento que RSA para verificar la firma. Es una variante de ElGamal.
Digital Signature Standard (DSS), publicación junio 2009,
http://csrc.nist.gov/publications/fips/fips186-3/fips_186-3.pdf

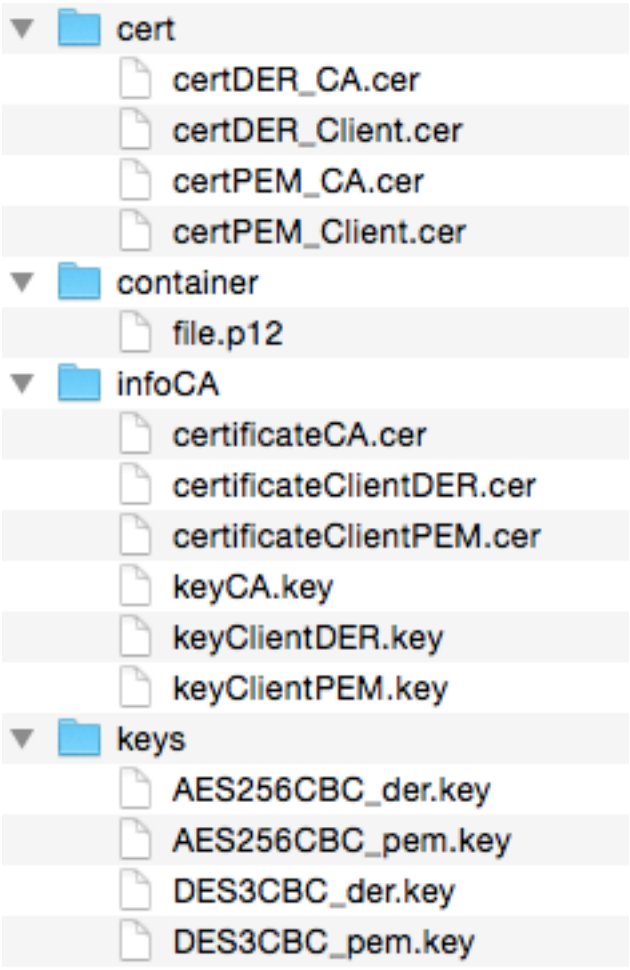
4.2.11 Pruebas unitarias

Tanto la generación de llaves privadas cifradas como los certificados digitales se construyen con base a los estándares definidos. Realizar pruebas con herramientas certificadas garantiza la correcta construcción de las llaves y certificados digitales.

OpenSSL son una serie de herramientas de código abierto de propósito general enfocadas a la criptografía. En los sistemas operativos de la familia *NIX se encuentran por default instalados ya que intervienen en múltiples tareas del sistema operativo.

Para la ejecución de la pruebas unitarias es necesario regenerar las llaves públicas y privadas, para eso es necesario ejecutar las pruebas existentes dentro la carpeta *test*.

Al ejecutar los JUnits se creara una estructura de archivos de la siguiente forma:



Se utilizara cada archivo en conjunto con openSSL para validar la correcta generación de llaves.

Validación de certificado digital

Despliegue completo de certificado con codificación DER.
$ openssl x509 -inform DER -in certDER_Client.cer -noout -text

Despliegue de subject de certificado con codificación PEM.
$ openssl x509 -in certPEM_Client.cer -noout -subject

Validación de llaves cifradas

Validación de notación ASN1 de una llave privada con 3DES y codificación DER.
$ openssl asn1parse -inform DER -in DES3CBC_der.key

Extracción de una llave privada cifrada con 3DES y codificación PEM, validación de notación ASN1 de llave privada.
$ openssl pkcs8 -in DES3CBC_pem.key -topk8 -nocrypt -out file.test
$ openssl asn1parse -in file.test

Validación OCSP

Para la conexión a un servidor de OCSP se requiere un certificado de la autoridad certificadora (certCA.cer), el certificado a validar (certCliente.cer) y la url del OCSP.

$ openssl ocsp -issuer certCA.cer -cert certCliente.cer -reqout ocsp.req
$ curl --data-binary @ocsp.req -o respuesta.rsp -v https://ocsp.com
$ openssl ocsp -respin respuesta.rsp -text -noverify >> output.txt

Validación de firma digital

Firma de documento file.test con llave privada cifrada y codificación PEM.
$ openssl sha1 -sign keyClientPEM.key -out data.rsa file.test

Extracción de la llave publica del certificado digital para realizar la validación de la firma contenida en data.rsa
$ openssl x509 -pubkey -noout -in certificateClientPEM.cer > pubkey.pem
$ openssl sha1 -verify pubkey.pem -signature data.rsa file.test

4.3 Módulo de mapeo

4.3.1 Definición del módulo

El control de los dispositivos conectados lo realiza el sistema operativo ya que selecciona los drivers adecuados para el correcto funcionamiento del dispositivo. Es por esta razón que se realizan consultas al sistema operativo para extraer la información de cada dispositivo conectado. Los principales tipos de dispositivos a utilizar son aquellos que sirven de almacenaje externo, tal es el caso de las memorias USB.

El modulo se encuentra definido base a una arquitectura Maven, dividido en una sección de código y una sección de pruebas unitarias. La estructura general del proyecto se muestra en la figura 4.7.

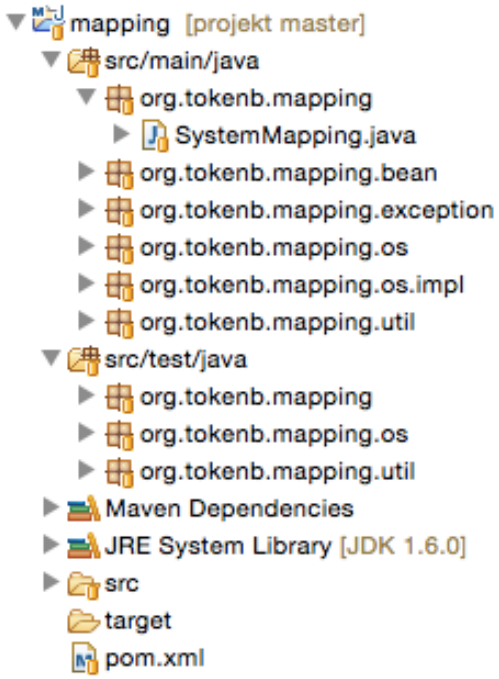


Figura 4.7: Estructura del módulo de mapeo

El código del módulo se encuentra dentro de la carpeta *src/main/java*. Las clases de acceso se encuentran definidas dentro del paquete *org.tokenb.mapping.*

4.3.2 Aspectos base del mapeo

El mapeo se refiere a una ubicación del dispositivo de almacenamiento en el sistema operativo.
Para el caso de Windows los mapeos se encuentran definidos por unidades identificadas por letras. Así cada letra estará asignada a un dispositivo de almacenamiento mientras esté conectado y en funcionamiento.

Para el caso de los sistemas *NIX se encuentran definidos por un punto de montaje, definido en un directorio del sistema.

Ambos sistemas operativos utilizan librerías o utilerías que ayudan a las tareas de mapeo simplificando así el uso de los dispositivos de almacenamiento. Para indicarle al sistema operativo que dispositivos USB se encuentran conectados se utiliza el descriptor del dispositivo.

Todos los dispositivos USB cuentan con descriptores (Figura 4.8) que proporcionan información tal como qué es el dispositivo, quién lo fabrica, qué versión de USB soporta, de cuántas formas puede configurarse. El acceso al descriptor se realiza de manera diferente entre sistemas operativos, pero los atributos asociados al dispositivo son constantes.

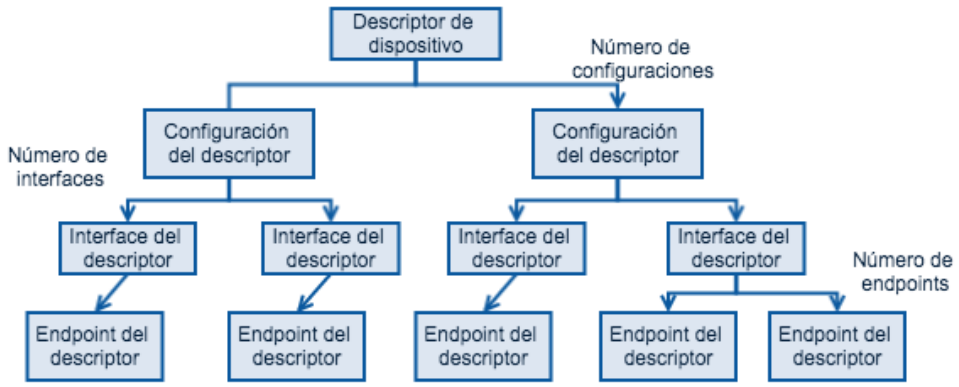


Figura 4.8: Interfaces del descriptor

4.3.3 Interfaces del módulo

La detección de dispositivos conectados se realiza mediante la clase *SystemMapping* que cuenta con métodos definidos para realizar consultas a los sistemas operativos por separado.

El método *getDevices* extrae todos los dispositivos USB conectados en una lista con objetos de tipo *device*. Esta consulta detecta que sistema operativo dispara la acción.

Para consultar cual es el sistema operativo local se encuentra definido el método *getOS* que regresa un entero. El catálogo de sistemas operativos se encuentra definido dentro de la clase *SystemMapping* como atributos estáticos para su consulta.

La extracción del descriptor se realiza mediante *DeviceDescriptor*, cuenta con un método llamado *findSerial* que ayuda a extraer el número serial del dispositivo seleccionado.

4.3.4 Descriptor OSX

El descriptor de OSX se obtiene mediante el comando *system_profiler* con el argumento *SPUSBDataType*.

Algunos de los atributos del descriptor son:
Product id, Vendor id, Version, Serial, Speed, Volumes, File System, Mount Point.
La figura 4.9 muestra los atributos más comunes de un descriptor.

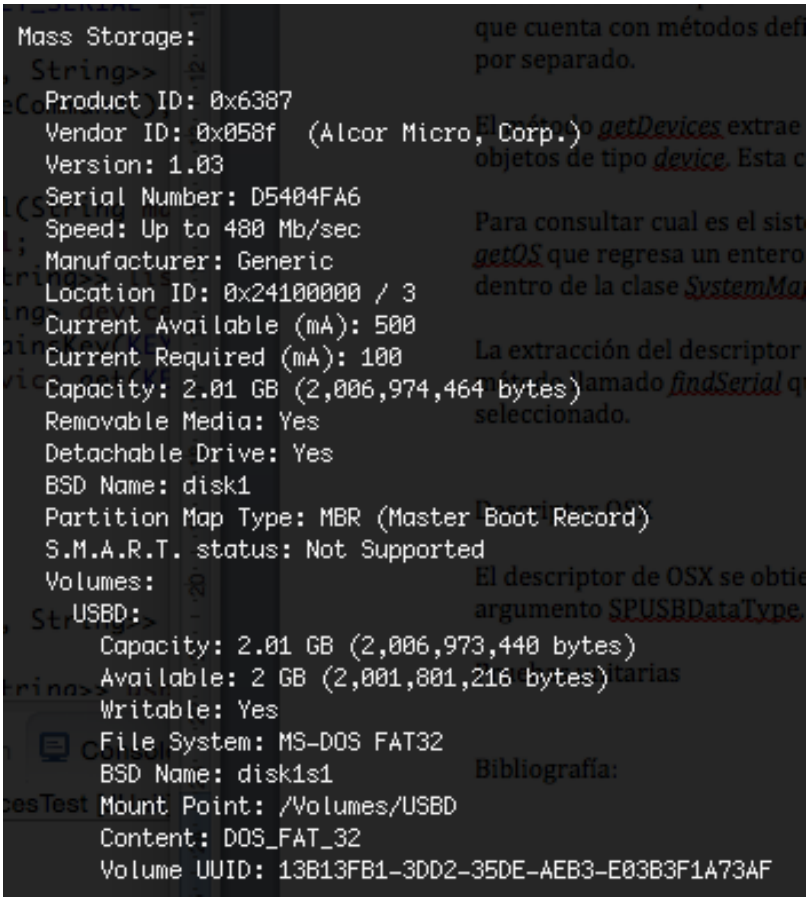


Figura 4.9: Atributos en descriptor OSX

El acceso al descriptor se realiza desde la clase OSX, ejecuta el comando y recupera todos los dispositivos conectados por puertos USB.

Los descriptores son almacenados dentro de una lista que asocian los atributos de cada dispositivo en una tabla tipo llave valor. En este punto todos los dispositivos USB son reconocidos ya sean de almacenamiento o para entrada de datos como es el mouse o el teclado.

Para descartar los dispositivos que no son de almacenamiento se validan los volúmenes asociados al dispositivo USB. Al contar con un volumen se puede acceder a la capacidad del dispositivo así como el sistema de archivos utilizado.

4.3.5 Acceso al descriptor en múltiples sistemas operativos

Dentro de las implementaciones para la obtención del descriptor se encuentra la asociada OSX que ejecuta un comando del sistema para la obtención de los atributos del descriptor. Obteniendo de esta forma el descriptor se requiere realiza una implementación por sistema operativo a utilizar, las principales ventajas son las de no requerir librerías de terceros para su utilización lo que hace más compatible la librería de mapeo.

Por otra parte la unificación del mapeo independientemente del sistema operativo se puede realizar con ayuda de la librería libusb.

4.3.6 Acceso a USB desde Java

Para realizar la comunicación con el dispositivo desde Java es utilizada la librería libusb, el problema radica en que esta librería se encuentra escrita en lenguaje C además aún se requiere una implementación de las acciones o pasos que se deben realizar para extraer el descriptor con ayuda de libusb.

La interacción de programas y librerías escritas en código C, C++ o ensamblador con aplicaciones escritas en Java que se ejecutan dentro de una Java Virtual Machine (JVM) se realiza mediante la interface Java Native Interface (JNI). La figura 4.10 muestra el diagrama de comunicación entre JNI.

Figura 4.10: Interacción Java y C

4.3.7 Implementación de librería JNI

Para realizar la implementación de la librería JNI que utilice usblib para la extracción del descriptor es necesario definir los métodos de acceso que utilizara JNI.

Todos los métodos de entrada deben llevar tanto los argumentos de entrada como el tipo de respuesta a retornar a la clase. Para realizar la definición de un objeto Java en código C se requiere pasar como argumento la clase a generar.

La construcción del objeto se realiza definiendo cada uno de los métodos de acceso de la clase para esto se utiliza *GetMethodID* una de las estructuras de JNI y sirve tanto para invocar un método de la clase como para definirlo. Los tipos de datos de los argumentos de cada método pueden ser de tipo primitivos, pero si el argumento es un objeto de Java se debe definir mediante el tipo firma.

Las firmas de cada objeto se definen en la tabla 4.1.

Tabla 4.1: Firmas de cada tipo

Tipo de firma	Tipo Java
Z	boolean
B	byte
C	char
S	short
I	int
J	long
F	float
D	double
L fully-qualified-class ;	fully-qualified-class
[type	type[]
(arg-types) ret-type	method type

Para el caso de un método
Long f(int n, String s, int[] arr)
Su tipo de firma seria:
(ILjava/lang/String;[I)J

Una vez definidos todos los métodos de la clase se instancia el objeto con la estructura *NewObject* pasando como argumentos los métodos definidos.

El retorno de información se realiza mediante un arreglo de objetos por lo que es necesario definir un arreglo y agregar cada objeto creado para almacenarlo con la estructura *SetObjectArrayElement*.

El tipo de dato del retorno de la función es una estructura *jobjectArray* que podrá comunicarse con Java.

Interface JNI

Las funciones definidas en C solo son accedidas por la máquina virtual en tiempo de ejecución. Para invocar la función en C mediante código Java se define un método de tipo nativo (*native*). Estos métodos no contienen implementación en Java, y al momento de ser compilados quedan definidos como interfaces de entrada.

Los métodos nativos son cargados por la máquina virtual con el método *System.loadLibrary*.

Generación de cabecera y librería nativa

Para generar las cabeceras a partir de los métodos nativos definidos en Java se utiliza el comando *javah*. La generación de las cabeceras y el código inicial se realiza dentro de la carpeta *cCode* mediante el script *makeJNIHeader.sh* con la opción *–H*. Esto creara una carpeta de nombre *genCode* con la cabecera (*.h*) y el programa inicial (*.c*)

Para la generación de la librería nativa se utiliza el script con la opción –L, obteniendo la librería *libinterfaceusb.jnilib*.

4.3.8 Pruebas unitarias

Las prueba unitarias se encuentran definidas dentro de las carpetas *src/test/java*.

El paquete *org.tokenb.mapping* contiene la clase de pruebas *SystemMappingTest* donde se accede a los descriptores del sistema operativo con base a comandos de sistema.

El paquete *org.tokenb.mapping.libusb* contiene la clase de pruebas *DescriptorJNITest* el cual accede a los descriptores de los dispositivos USB conectados utilizando libusb mediante JNI.

4.4 Módulo de dispositivo

4.4.1 Definición del módulo

El módulo de dispositivo es el encargado de realizar la comunicación de alto nivel y expone la funcionalidad del dispositivo de cifrado mediante la librería *device*.

Con ayuda de la librería de *mapeo* se selecciona uno de los dispositivos USB conectados para crear el contenedor de llaves. Las llaves tanto pública como la privada son copiadas al dispositivo y utilizando la librería de *seguridad* se genera una firma del dispositivo para poder utilizarlo con fines de autenticación o firma de documentos. La conexión entre librerías se muestra en la figura 4.11.

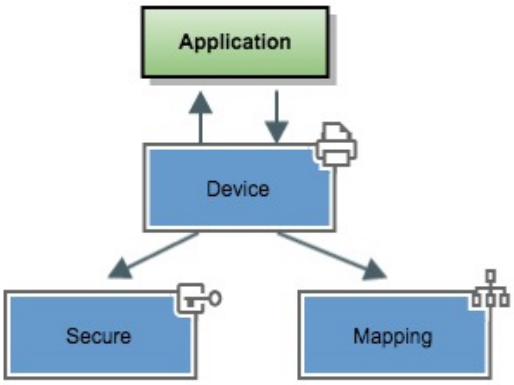


Figura 4.11: Conexión de librería de dispositivo

El modulo se encuentra definido base a una arquitectura Maven, dividido en una sección de código y una sección de pruebas unitarias. La estructura general del proyecto se muestra en la figura 4.12.

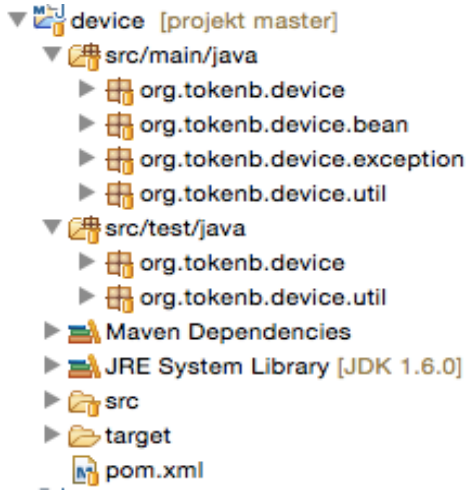


Figura 4.12: Estructura del módulo de dispositivo

El código del módulo se encuentra definido dentro de la carpeta *src/main/java*. Las clases de acceso se encuentran definidas dentro del paquete *org.tokenb.device*.

4.4.2 Token de seguridad

La autenticidad de identidad proporcionada por medios electrónicos no garantiza que la persona exista o que la información proporcionada sea correcta. En México el principal documento de identificación es la credencial para votar IFE, se encuentra en constante actualización agregando mecanismos de seguridad contra la falsificación. Pero este documento solamente sirve a nivel para validación de identidad a nivel visual.

Por otra parte el SAT comenzó a realizar un padrón de todos sus contribuyentes registrados, mismos que ya se encuentran registrados ante el IFE, con el fin de garantizar la identidad de la persona. La identidad se asocia obteniendo las huellas dactilares, captura de iris, fotografía de frente así como la firma autógrafa. Con este proceso se asocia un juego de llaves (pública y privada), donde la llave pública se encuentra dentro de un certificado digital con la información de la persona y un número de serie de la autoridad certificadora. El certificado digital es público pero la llave privada cifrada es intransferible por lo que su uso es responsabilidad del dueño de la llave.

La asociación de un documento físico visual con un juego de llaves para tener un único documento de identidad aún no se encuentra implementado en México. En países como Alemania, Bélgica o España ya cuentan con Documentos de Identidad Electrónica (DNIe) donde se almacena información electrónica vía RFID[16] para

[16] RFID Radio Frequency Identification tecnología para transmitir y almacenar información de forma remota. En modo pasivo no requieren alimentación. Se basa en el estándar ISO 14443.

interactuar con aplicaciones gubernamentales pero también contiene información impresa para autenticación visual.

Al utilizar tecnologías específicas como es el RFID o NFC[17] simplifican la comunicación entre dispositivos haciendo su uso aún más simple entre personas, el problema radica en las interfaces para interpretar la comunicación. Estas interfaces o lectoras deben estar presentes en todos los lugares donde se requiera utilizar, ya sea para autenticación en un portal o firmar un documento.

Para incentivar la inclusión de mecanismos de llaves asimétricas es necesarios utilizar dispositivos costeables y herramientas para su gestión e integración con aplicaciones existentes. El dispositivo físico que ayuda en estas labores se denomina Token de Seguridad.

Un token de seguridad es un dispositivo físico (tarjeta, generador de números, dispositivo USB) que proporcionan un servicio de autenticación en computadoras basado en llaves criptográficas.

La utilización de tokens USB facilitan la comunicación entre dispositivos ya que un estándar en los puertos de las computaras permite encontrar por lo menos un puerto USB ya sea en versión 1.0 o 2.0.

4.4.3 Generación de token USB

La construcción de un token USB basado en llaves asimétricas comienza con la extracción de las características del dispositivo, esto se realiza con el descriptor del dispositivo USB que proporciona la huella del dispositivo.

La huella del dispositivo es firmada con una llave privada, como se muestra en la figura 4.13.
El módulo de seguridad cuenta con implementaciones para poder leer codificaciones DER y PEM así como soporte algoritmos de cifrado simétrico de llaves

Para mantener compatibilidad entre autoridades certificadoras y reutilizar las llaves cuenta con una implementación para interpretar las llaves generadas por SAT. Al utilizar una llave proveniente de una autoridad certificadora de confianza se provee el respaldo de una institución ante una controversia legal.

ISO 14443, ultima modificación julio 2014, http://es.wikipedia.org/wiki/ISO_14443

[17] NFC Near Field Communications es un subconjunto de RFID su alcance es menor a 10 cm.
NFC, ultima actualización marzo 2015, http://es.wikipedia.org/wiki/Near_field_communication

La generación de llaves privadas y certificados digitales de una autoridad certificadora con OpenSSL también se encuentra soportado por el módulo de seguridad.

En caso de no contar con certificados digitales de una autoridad certificadora dentro del módulo de seguridad se encuentran interfaces para la implementación de una autoridad certificadora y así utilizar certificados propios.

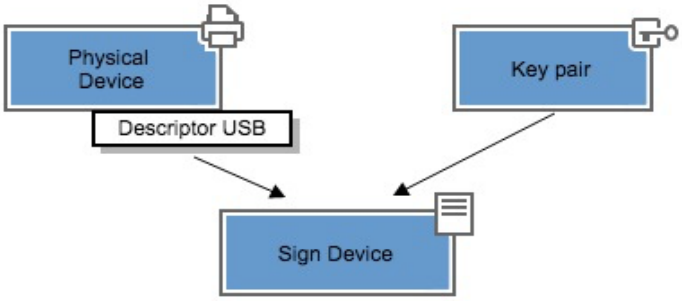


Figura 4.13: Firma del dispositivo.

La firma correspondiente al descriptor del dispositivo puede ser validada con la llave pública ubicada dentro del certificado digital. Es almacenada en un archivo con la extensión .token dentro del contenedor del dispositivo.

El dispositivo USB a utilizar es una memoria de almacenamiento con interface USB.
Las memorias al ser de bajo costo pueden tener la funcionalidad de almacenar información y ser utilizadas como token USB.

El sistema de archivos de la memoria USB es utilizado como contenedor del juego de llaves que servirán como insumos del token. El contenedor es una carpeta oculta para el usuario, su intención es agrupar los posibles tokens a almacenar en una misma memoria USB. La figura 4.14 muestra la estructura para contenedor de llaves y certificados
.
Una misma memoria USB puede generar diferentes tokens asociados a diferentes llaves para ser utilizados por separado.

El contenedor agrupa las llaves por un alias o *tokenapp*, esto ayuda a identificar las llaves que se encuentran dentro de un token y así conocer para que pueden ser utilizadas (perfiles de seguridad).

La intención del contenedor es solo de almacenaje centralizado en el dispositivo, no pretende ser utilizado como contenedor de seguridad. Las llaves privadas se encuentran cifradas por lo que su acceso no es directo y requieren de la contraseña para acceder a la llave. Si es requerido un almacenamiento seguro de las llaves en el dispositivo puede ser utilizado PKCS #12. La implementación se encuentra en el módulo de seguridad.

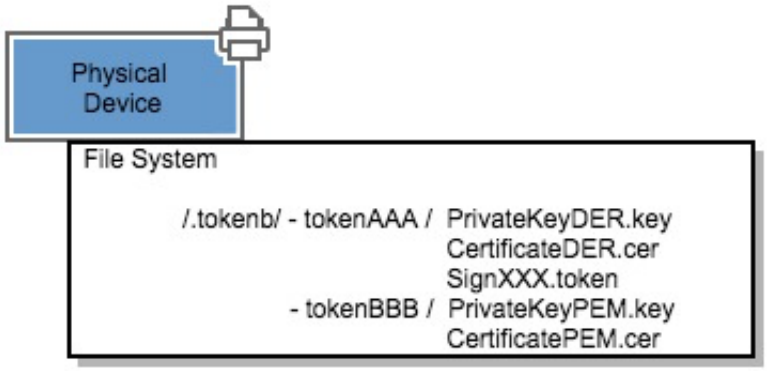


Figura 4.14: Sistema de archivos del dispositivo

Para la generación de un token USB se utiliza la clase *CreateUsbToken*, con el método *setTokenApp* se define el identificador para agrupar los elementos del token, se puede utilizar una descripción de catálogo o un simple UUID. Por default se encuentra definido el identificador *personal*, en caso de no definir alguno.

El constructor de la clase requiere como argumento un objeto de tipo *TokenB*, este objeto contiene el dispositivo a utilizar como token, la ruta del certificado digital, la ruta de la llave privada y la contraseña de la llave privada. La contraseña es requerida para extraer la llave privada y poder realizar la firma del dispositivo, también es utilizada en caso de realizar un almacenamiento seguro con PKCS #12.
Para crear el token se utiliza el método *make*, que realiza la validación de las llaves y extrae el descriptor del dispositivo seleccionado para posteriormente generar la firma del dispositivo y almacenarla en su contenedor.

4.4.4 Consumo de token

La identificación de los dispositivos conectados es la primera etapa para consumir un token, no todos los dispositivos conectados son tokens.

La primera validación se realiza mediante el *tokenapp*, que busca dentro del contenedor el identificador de agrupación de las llaves y la firma. Al encontrar un identificador solo garantiza la asociación con las llaves, pero que sucedería si un contenedor se realizara cambiando el nombre mediante un el comando *mv*.

La firma del dispositivo contiene el identificador de agrupación por lo que sí es modificado intencionalmente la firma contenida en el dispositivo no sería válida.

En la implementación del cliente para la autenticación esta firma es almacenada del lado del servidor ya que es comparada para iniciar sesión en un aplicativo.

Por último se realiza la validación del juego de llaves para comprobar su concordancia. De ser diferentes el token podría estar alterado por lo que no podría ser utilizado.

La recuperación de los tokens se realiza con la clase *UsbToken* que utiliza el método *setTokenApp* para definir el identificador del token.

Para realizar la consulta de tokens activos se utiliza el método *retrieveDevices* que devuelve una lista de objetos de tipo *TokenB*.

4.4.5 Pruebas unitarias

Se encuentran definidas dentro de la carpeta *src/test/java* bajo el paquete *org.tokenb.device*.

Las pruebas contemplan la generación de un token USB a partir de un dispositivo de almacenamiento conectado, por lo que de no definirse un dispositivo USB existente en el sistema operativo la prueba no funcionara. Las llaves utilizadas para esta prueba son extraídas del proyecto de seguridad por lo que es necesario primero correr las pruebas en dicho proyecto para que puedan ser leídos los archivos del certificado digital y la llave privada cifrada.

La clase de prueba *CreateUsbTokenTest* genera dos tokens en el mismo dispositivo USB pero con diferentes identificadores.

La prueba para consumir el token debe ser ejecutada posterior a la generación del dispositivo ya que extrae una lista de tokens a partir de un identificador. La prueba de generación crea en un solo dispositivo dos tokens, en esta prueba recupera cada uno de acuerdo a su identificador.

La clase de prueba *UsbTokenTest* crea listas con los tokens encontrados en el sistema operativo de acuerdo al identificador definido.

Capítulo 5

Implementación web

5.1 Definición del aplicativo

La intención del aplicativo es mostrar una implementación web para autenticación mediante un token USB. Esta implementación puede ser modificada para firmar documentos de electrónicos así como para su validación.

La aplicación se encuentra dividida en dos módulos, *web* donde se encuentra la funcionalidad que se ejecuta del lado del servidor para validación de acceso mediante llaves y *Applet* que se ejecuta del lado cliente para interactuar con el token USB. La figura 5.1 muestra la comunicación entre los módulos.

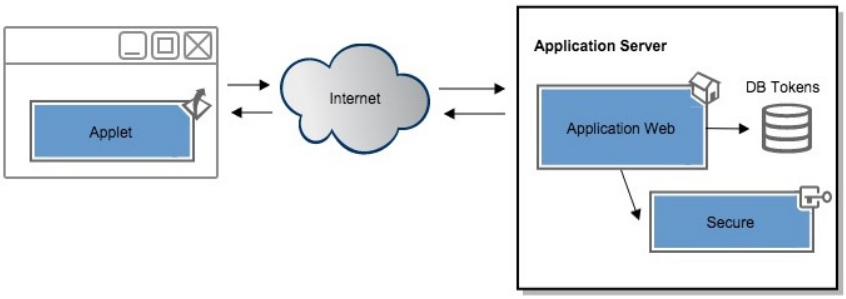


Figura 5.1: Comunicación entre módulos

El modulo web se encuentra definido base a una arquitectura Maven, dividido en una sección con código que se ejecuta del lado del servidor, recursos de publicación y pruebas unitarias. El código se encuentra dentro de la carpeta *src/main/java* y se encuentra empaquetado dentro de *org.tokenb.web* como lo muestra la figura 5.2.

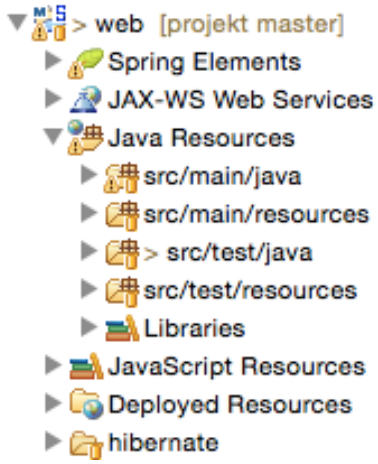


Figura 5.2: Definición de código servidor

La sección con elementos definidos por el servidor pero interpretados por el cliente como recursos de publicación web (imágenes, páginas, scripts y estilos) y configuración de la aplicación son definidos dentro de la carpeta *src* tal como lo muestra la figura 5.3.

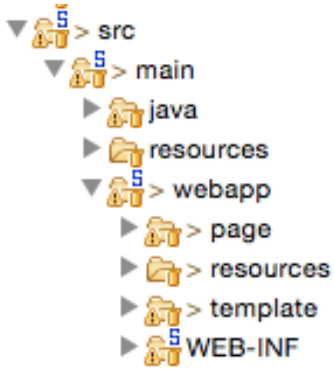


Figura 5.3: Definición de elementos cliente.

5.2 Arquitectura

Al ser un proyecto web la comunicación con los clientes así como las vista son gestionadas por el servidor, esto implica una correcta segmentación de funcionalidades o capas que agrupen comportamientos específicos de la aplicación. La arquitectura se basa en un patrón de diseño conocido como MVC que separa el modelo de datos, la vista y el controlador para funcionar de manera independiente. Esto ayuda al momento de realizar modificaciones y no alterar el funcionamiento de cada componente.

5.3 Configuración de frameworks

La configuración general de la aplicación se encuentra definida dentro del archivo *web.xml* que utiliza el servidor de aplicaciones para registrar la aplicación dentro de su espacio de aplicaciones.

Spring MVC define en este archivo el comportamiento de las peticiones web y como se deberán canalizar utilizando un *dispatcher*. Para indicar las reglas que utilizara MVC para resolver las vistas se utiliza el archivo *dispatcher-servlet.xml*.

Spring Framework es utilizado para interactuar con las operaciones del motor de base de datos, primero se debe cargar el contexto de los objetos a utilizar antes de que sean utilizados por la aplicación, es utilizado *ContextLoaderListener* que genera una instancia en el contexto de Spring, su definición se encuentra dentro del archivo *applicationContext.xml*.

La configuración para interactuar con el motor de base de datos se encuentra definida dentro del archivo *hibernate.xml.*

Spring JDBC es utilizado para definir un objeto de conexión a base de datos denominado *DriverManagerDataSource* asociado a un driver de comunicación donde se indica IP del servidor, puerto, nombre de base de datos, usuario y contraseña.
Cada tabla de base de datos se encuentra asociada con un objeto de programación (*bean*), los beans son construidos por Hibernate a partir del modelo de datos. Cuando inicia el contexto de Spring son definidos estos objetos como traductores para realizar operaciones en base de datos. El objeto *LocalContainerEntityManagerFactoryBean* es el encargado de realizar la instancia de los objetos de Hibernate.

El acceso a datos se encuentra definido mediante Hibernate utilizando objetos de programación, pero las operaciones a base de datos se tendrían que definir en lenguaje SQL. Spring JPA encapsula la funcionalidad de consultas de SQL utilizando repositorios que son construidos a partir de interfaces de programación. Las consultas son realizadas con base a los objetos de programación a partir de métodos definidos en la interface del repositorio. El manejador de operaciones a base de datos se instancia mediante *JpaTransactionManager* y se indica que interfaces deberá utilizar para su repositorio de acceso a datos mediante *jpa:repositories*.

Al iniciar el servidor de aplicaciones es iniciada una máquina virtual que ejecuta la aplicación Java. Cada máquina virtual cuenta con un proveedor de servicios de seguridad autorizado, el archivo *bouncy-castle.xml* es utilizado para agregar al proveedor de seguridad Bouncy Castle al contenedor de proveedores seguros.

El archivo *dispatcher-servlet.xml* es utilizado para definir el comportamiento de peticiones al servidor. Las peticiones pueden atender a llamadas de acceso a datos o vistas. Las vistas son controladas por *mvc-config.xml* con ayuda de Tiles para armar cada vista, la configuración se realiza mediante la clase *TilesConfigurer* y las reglas de

cómo se deberán resolver las vistas se especifica mediante *ContentNegotiatingViewResolver*.

Las vistas pueden estar compuestas por elementos modulares reutilizables como el header, footer o el menú de la aplicación. La composición de las vistas se realiza mediante el archivo *tiles.xml* donde se asocia cada elemento de la vista con un JSP de código para armar una página web. Cada definición de vista está identificado por un nombre que utiliza Spring MVC para construir la vista final de la página.

5.4 Modelo de datos

Se construyó un modelo de datos simple para almacenar los tokens válidos para el uso de inicio de sesión en aplicaciones. Tiene la capacidad de poder ser utilizado como modelo central en múltiples aplicaciones ya que con ayuda del catálogo *TokenApp* se pueden agregar aplicaciones asociadas a perfiles del catálogo *Profile*.

El usuario se encuentra definido en *UserToken* donde es almacenada la información del descriptor USB y la firma que genero la llave privada. Cada *UserToken* lleva asociado un *TokenApp* referente al catálogo de aplicaciones y un perfil. Para validar su vigencia se cuenta con una fecha de a partir que es válido el token y una fecha de expiración.

Para descifrar los mensajes provenientes del cliente es utilizada la llave publica que se encuentra contenida en el certificado digital. Es utilizada la tabla *Certificate* para almacenar el certificado digital, un *UserToken* puede tener asociados múltiples certificados digitales ya sea por perfiles en diferentes aplicaciones o por tener diferentes certificados digitales provenientes de múltiples autoridades certificadoras.

En el modelo de datos cada vez que se realiza una autenticación se genera un token que es válido en solo una ocasión y es almacenado dentro de la tabla *Authentication*. Sí es reutilizado la aplicación denegara el acceso ya que se considera expirado.

El diagrama entidad relación se muestra en la figura 5.4.

5.5 Recursos cliente

En el desarrollo de aplicaciones web parte del código implementado es ejecutado del lado del servidor mediante el navegador.

Los elementos de la página son definidos en HTML, dentro de la carpeta *page* se encuentran definidas las secciones de página que utilizara Tiles para armar la página completa.
Las plantillas se encuentran dentro de la carpeta *page/template* son utilizadas como base para la página de bienvenida y autenticación. La plantilla *master.jsp* se utiliza

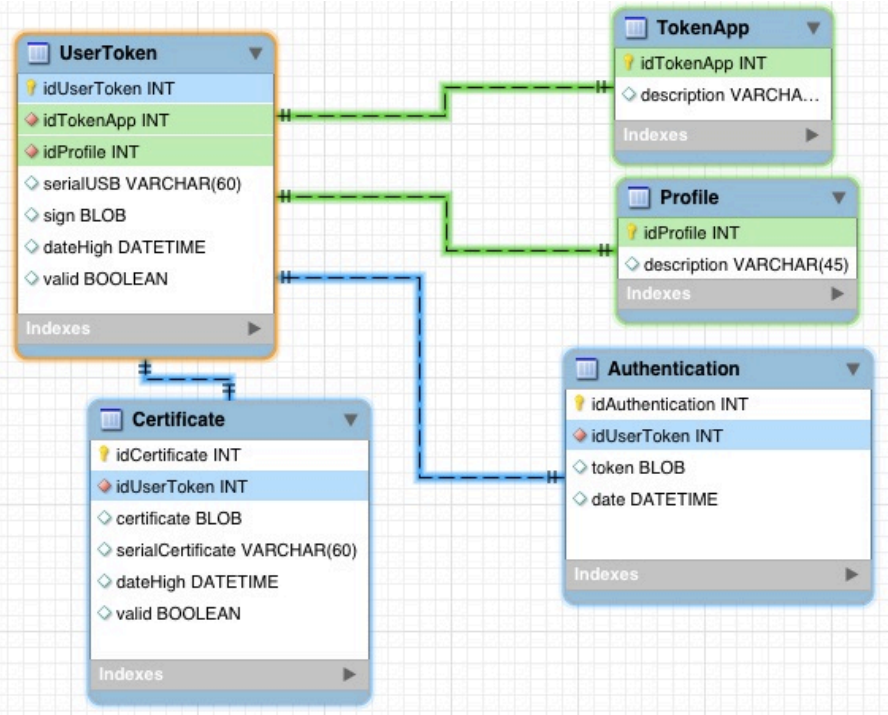


Figura 5.4: Modelo de datos

como base en toda la aplicación ya que se encarga de cargar los estilos, JavaScripts y los módulos de AngularJS.

Al tener la vista del lado del cliente no todo el control está del lado del servidor es por eso que con ayuda de AngularJS se separa la vista del control. Cada vista en HTML lleva asociado un control en JavaScript que con ayuda de AngularJS es direccionado para tener un control por vista.

Las aplicaciones web se caracterizan por desplegarse diferente en diversos navegadores y dispositivos, esto se corrige con la ayuda del framework Bootstrap. Se utilizan los estilos y JavaScripts de Bootstrap para homologar las vistas en los dispositivos, además de proporcionar un desarrollo ágil de interfaces graficas al tener componentes ya definidos.

5.6 Métodos REST

La aplicación cuenta con métodos REST definidos para realizar las peticiones de autenticación así como para resolver las vistas.

Dentro del paquete *org.tokenb.web.controller* se encuentran definidos los mapeos REST.

La clase de control *StartControl* por default intercepta la petición GET dirigida a / y es re direccionada a la página de inicio */start.*

Los errores se encuentran definidos dentro de la clase *ErrorMessage* que intercepta las peticiones de tipo */error*, en caso de realizar una petición a una página que requiera autenticación y no tenga permisos de acceso es re direccionada a */error/nosession*.

La clase *MainPage* contiene el acceso privado al sitio, la peticiones GET */sesión/mainpage* valida la existencia de una sesión activa para poder acceder. En caso de existir despliega la página privada de inicio sino re direcciona a */error/nosession*.

Para realizar el acceso a la aplicación mediante token es utilizada la clase *Authenticate*. Todas las peticiones de la clase Authenticate tienen como petición base /auth.

La petición GET */auth* devuelve la vista de autenticación que contiene el Applet para interactuar con el token USB.

Al comenzar la sesión el Applet envía un objeto JSON con las características del token que validara la firma almacenada en base de datos. Sí el token se encuentra registrado se envía una cadena UUID que deberá ser firmada con la llave privada. El inicio de sesión se realiza con el método POST /auth/start_token.

La validación de la cadena firmada UUID es realizada con la petición POST /auth/check donde es utilizado el certificado digital del usuario para verificar la firma enviada.
Sí la cadena UUID enviada es igual a la cadena contenida de la firma se genera un identificador de llave en conjunto con una llave de 512 caracteres para que el aplicativo inicie sesión desde JavaScript.

Para acceder al aplicativo se accede a la petición POST /auth/start_session donde es re direccionado al sitio base con sesión /mainpage.

La autenticación almacena cada una de las llaves de acceso temporal para descartar los accesos múltiples no autorizados asociados a una misma llave temporal.

5.7 Comunicación con token USB en cliente

La integración del token USB con un aplicativo web implica la construcción de un cliente que puede acceder de manera nativa al sistema, en este caso a un dispositivo USB conectado.

Los navegadores web interpretan, ejecutan y despliegan información mediante HTML, hojas de estilos y JavaScript para desplegar una página web. El navegador web directamente no accede a dispositivos disponibles en el sistema operativo, algunas aplicaciones como Flash acceden a dispositivos mediante plugins que se instalan en el navegador esto implica la instalación de plugins para cada tipo de navegador.

Otra forma de realizar la ejecución de aplicaciones a través del navegador web es mediante Applets. Los Applets interactúan con el navegador web mediante un plugin de Java, que ejecuta la aplicación en un contenedor restringido dentro del contexto del navegador web. Los Applets al estar dentro del contexto del navegador web pueden interactuar con elementos del DOM de la página web de forma bidireccional. Esto se traduce en ejecutar una aplicación de alto nivel con comunicación directa al DOM de la página web, ampliando en gran medida la funcionalidad de una aplicación web.

Debido a las funcionalidades adicionales proporcionadas por el Applet se utilizó para la conexión con los dispositivos USB como para el descifrado de la llave privada.
El proyecto Applet aloja el código del componente *Applet*, se encuentra definido base a una arquitectura Maven y utiliza las dependencias de Bouncy Castle así como de los módulos de secure y mapping para búsqueda de tokens y validación de llaves.

El Applet se comunica con base en el protocolo HTTP que consume los servicios de autenticación */auth* REST del servidor para establecer una sesión en la aplicación web. El control se encuentra implementado en la clase *ControlToken* que es ejecutada solo si es encontrado un token valido conectado.

La Figura 5.5 muestra el esquema de comunicación entre el Applet y la aplicación web en cada etapa del cliente con el servidor.

5.8 Implicaciones de usar Applet

Debido a que el Applet utiliza una máquina virtual de Java para poder ser ejecutado, debe existir una instalación de JRE de Java en la maquina cliente para su correcto funcionamiento.

A partir de la versión 1.7 de Java se incrementó el nivel de seguridad mínimo al momento de ejecutar Applets desde un navegador web. Las nuevas políticas de Java restringen la longitud de las llaves privadas esto se corrige actualizando los archivos de políticas local_policy.jar y US_export_policy.jar por las versiones extendidas proporcionadas por Java. Estas políticas deben sustituir a las políticas default contenidas dentro del directorio <java-home>/lib/security.

Una forma de validar que el código no ha sido codificado es firmando todo el contenido del Applet con un certificado digital de confianza, con esto se garantiza que no se ejecutara código diferente al publicado por el desarrollador.

Adicionalmente se valida el certificado digital con el que fue firmado el Applet, esto con la intención de proporcionar aún más confianza de uso a los usuarios debido a que esta certificación es por parte de una autoridad certificadora reconocida.

5.9 Alternativa a Applets

Una de las opciones de porque utilizar un Applet es por reutilización de código ya que se encuentran librerías de Java que ayudan en el desarrollo de una aplicación. Con la evolución de JavaScript el desarrollo de nuevas librerías de propósito general se encuentra en auge, un ejemplo de esto es JQuery y las múltiples aplicaciones que se han desarrollado entorno a esta librería. JavaScript con ayuda de HTML5 ahora pueden interactuar de una forma más activa con las aplicaciones web, pudiendo sustituir los Applets, la única limitante actual seria que no todos los navegadores web soportan HTML5 pero eso solo es cuestión de meses.

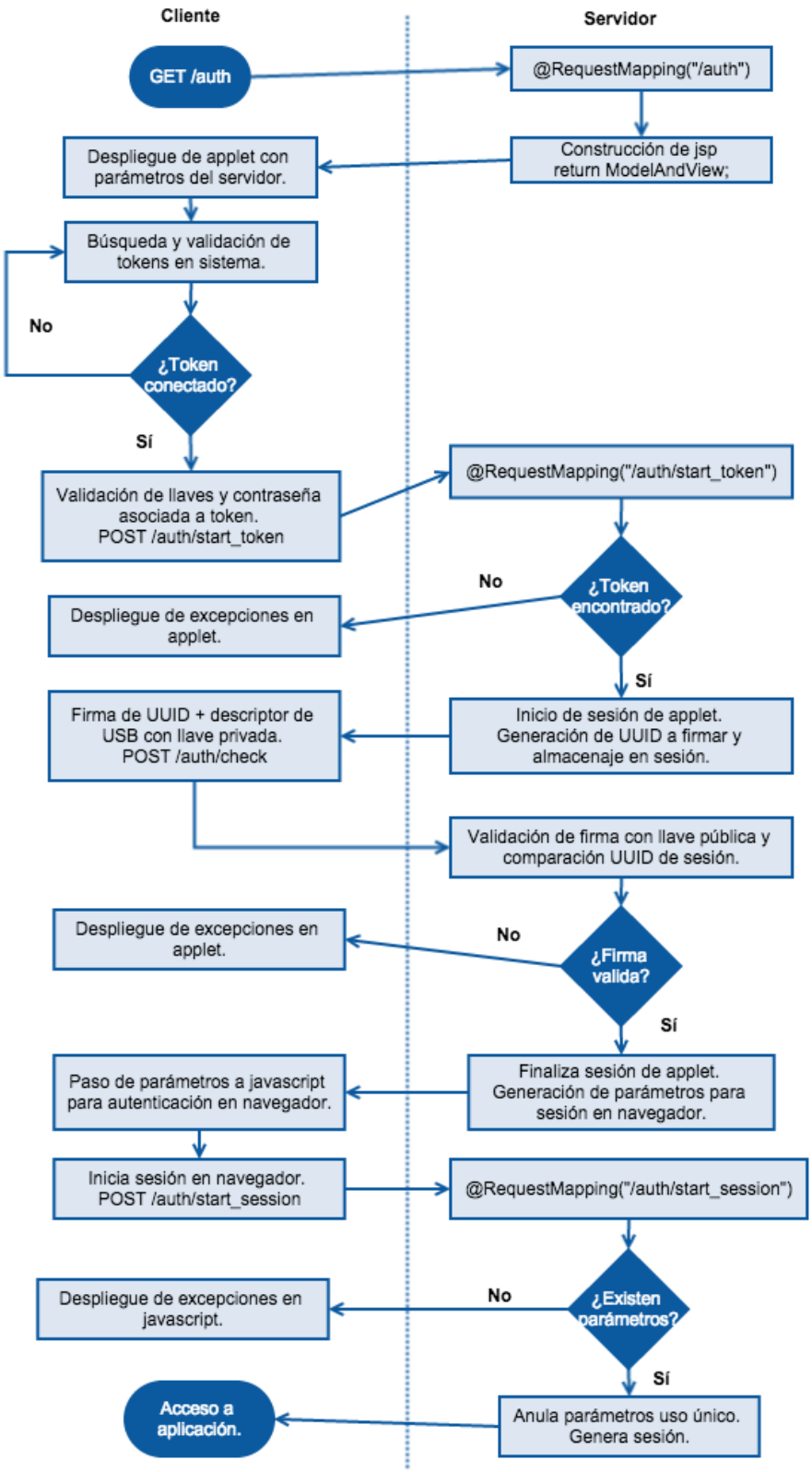


Figura 5.5: Esquema de autenticación Cliente-Servidor

Conclusiones y comentarios finales

La construcción de un framework para interactuar con llaves asimétricas surge a razón de poder utilizar las llaves privadas que proporciona el SAT y reutilizar los beneficios que ofrece una autoridad certificadora de confianza. Aunado a esto existía la posibilidad de poder integrar diferentes autoridades certificadoras y que trabajaran de manera transparente para el usuario.

El reto principal era desarrollar un estándar de comunicación que fuera compatible con diversos aplicativos, ya sea para aplicaciones de uso local (standalone), web o móviles.

El uso de código libre para desarrollo de software facilita la implementación y permite la estandarización en la comunicación, ayudando así en la integración de aplicaciones existentes como en desarrollos de software que apenas comienzan.

Al generar una serie de implementaciones es requerido realizar la documentación de los módulos implementados así como de su utilización, por esta razón que cada módulo no solo cuenta con una documentación sino también una serie de pruebas unitarias para que el desarrollador comprenda a detalle el funcionamiento básico de cada módulo.

Cada módulo se encuentra organizado (empaquetado) separando las implementaciones de acceso, las excepciones, los objetos de dominio, las interfaces y sus implementaciones así como las utilerías. Esto ofrece un mejor entendimiento del código para su modificación.

Los módulos actuales contienen implementaciones para interactuar con llaves privadas cifradas, certificados digitales, acceso a dispositivos USB, generación de tokens de seguridad, firma de documentos digitales y validación de firmas.

Constantemente los módulos son actualizados para incluir nueva funcionalidad o mejor aún es agregado un nuevo módulo para ofrecer nueva funcionalidad.

La integración de los módulos en un aplicativo web ofrece un completo ejemplo de funcionalidad aplicada en autenticación de sitios web utilizando un token de seguridad. Esta implementación puede ser tomada como base para el desarrollo de nuevas funcionalidades como firma electrónica de documentos mancomunados en aplicaciones web.

La prueba de concepto no solo muestra las implementaciones desarrolladas para el framework sino también los estándares de desarrollo de software utilizados para

aplicaciones web, proporcionando así una base para entender los conceptos básicos de comunicación, arquitectura, seguridad, modelado de datos y acceso a dispositivos.

El framework hace uso de librerías de software libre que ayudan a simplificar tareas, la mayoría de estas librerías son utilizadas en aplicaciones de uso empresarial por lo que su estabilidad se encuentra probada. Además de que proveen actualizaciones para corregir problemas de estabilidad así como de seguridad, por lo que es importante de que al utilizar este tipo de librerías se debe estar al tanto de las nuevas actualizaciones existentes.

Al utilizar software libre existe la ventaja de poder ver el código y conocer cómo se encuentra implementado con la posibilidad de poder modificarlo y distribuirlo pero respetando la licencia de uso.

Al reunir una serie de módulos de acceso libre enfocados a interactuar con llaves asimétricas se facilita la integración en aplicaciones de uso cotidiano pudiendo así no solo ofrecer una interacción más simple con los usuarios, sino además poder ofrecer nuevas herramientas para interactuar con documentos digitales de una forma completamente digital sin la necesidad de imprimir papel para que sea firmado de forma autógrafa.

Conforme la tecnología va tomando terreno en las tareas cotidianas de nuestras vidas la dependencia con los dispositivos personales aumenta en gran medida y se refleja en cada aspecto de nuestras vidas. Es por esta razón que un módulo del framework se encarga de establecer comunicación con los dispositivos y así poder interactuar de manera simple entre dispositivos.

La inclusión de nuevas tecnologías para dar mayor campo de acción han obligado a agregar más comunicación no solo USB sino también con dispositivos inalámbricos como NFC o dejar a un lado los Applets y utilizar únicamente JavaScript para el manejo de llaves.

No solo el hardware ha evolucionado, también el software se reinventa para facilitar el uso de computadoras personales. Esto beneficia tanto a usuarios como a desarrolladores, estos últimos abriendo amplias posibilidades para realizar implementaciones a soluciones aun no existentes o mejorar las actuales.

El framework pretende dar solución a ciertos aspectos de la criptografía enfocados a ser utilizados en las aplicaciones de uso común brindando una comunicación más segura, simple y eficiente. Nuevos módulos se encuentran en desarrollo para ser integrados al framework, con el fin de ofrecer una solución simple pero robusta para poder integrar en cualquier aplicación una implementación de llave asimétrica.

Apéndices

Clonar proyecto de repositorio

Los proyectos desarrollados en esta tesis se encuentran almacenados en un repositorio Git para acceso público.

Para descargar los proyectos se debe acceder a la página.
http://www.kypmmhoff.com:8088/group-tokenb/tokenb

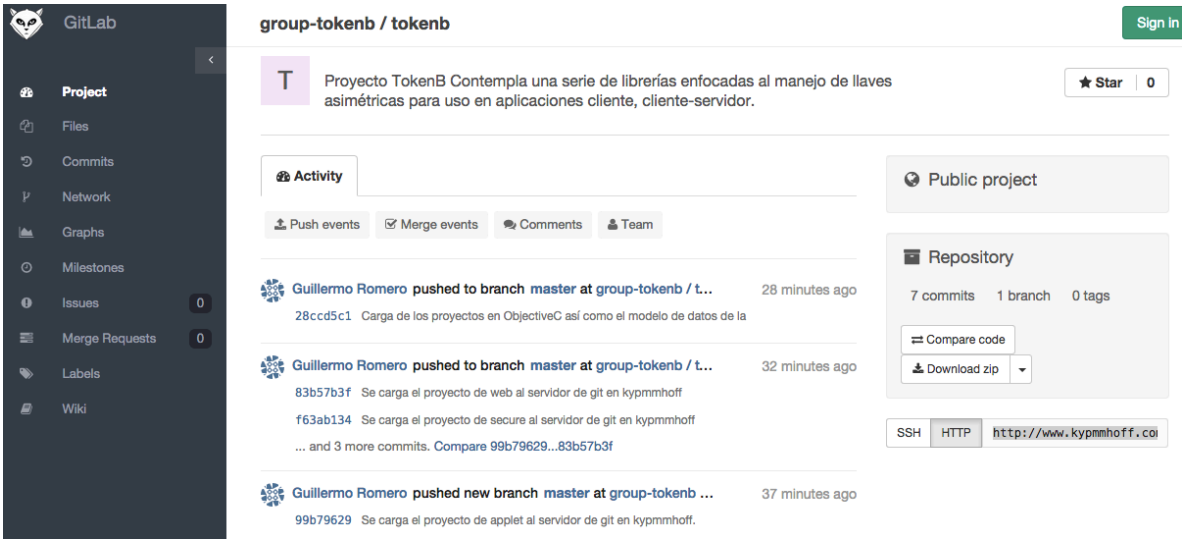


Figura 1: Acceso al repositorio

Dentro del repositorio se puede navegar dentro del código para consultar piezas de código. También se pueden descargar los proyectos para agregarlos dentro de una implementación. *Figura 2.*

Para descargar todos los proyectos basta con acceder a la página principal y oprimir el botón de *Download*, o acceder a cada uno de los proyectos y realizar el mismo procedimiento.

Los requerimientos básicos para utilizar los proyectos Java son:
- JDK 1.8
- Maven 3.2
- Tomcat 8
- MySQL 5.6

Para los proyectos en Objective C los requerimientos básicos son:
- XCode 6.3
- Libusb 1.0

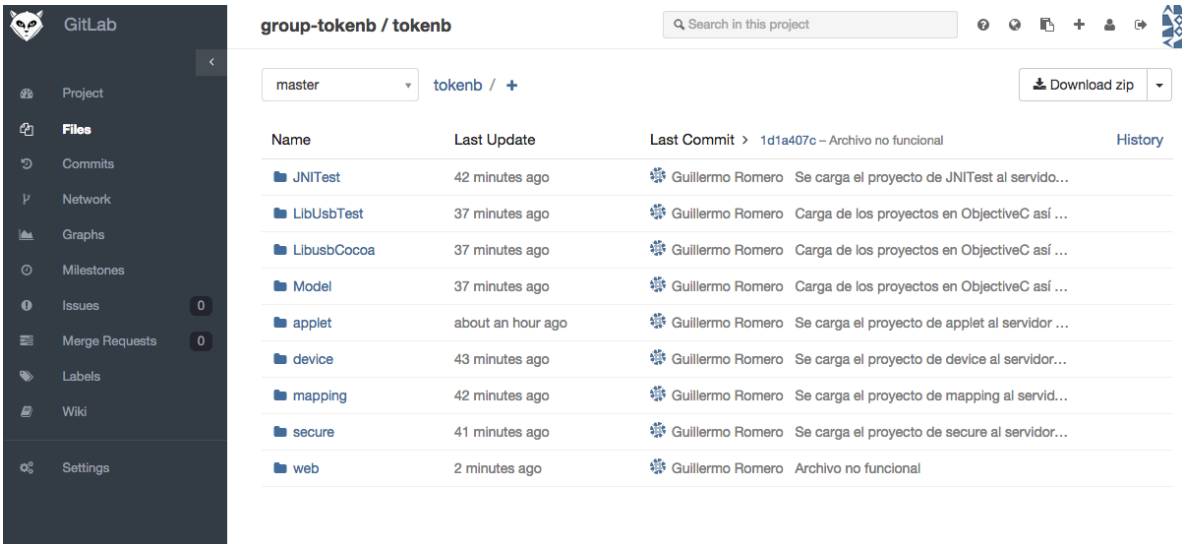


Figura 2: Proyectos en el repositorio

Tablas de permutación DES

(a) Initial Permutation (IP)

58	50	42	34	26	18	10	2
60	52	44	36	28	20	12	4
62	54	46	38	30	22	14	6
64	56	48	40	32	24	16	8
57	49	41	33	25	17	9	1
59	51	43	35	27	19	11	3
61	53	45	37	29	21	13	5
63	55	47	39	31	23	15	7

(b) Inverse Initial Permutation (IP^{-1})

40	8	48	16	56	24	64	32
39	7	47	15	55	23	63	31
38	6	46	14	54	22	62	30
37	5	45	13	53	21	61	29
36	4	44	12	52	20	60	28
35	3	43	11	51	19	59	27
34	2	42	10	50	18	58	26
33	1	41	9	49	17	57	25

(c) Expansion Permutation (E)

32	1	2	3	4	5
4	5	6	7	8	9
8	9	10	11	12	13
12	13	14	15	16	17
16	17	18	19	20	21
20	21	22	23	24	25
24	25	26	27	28	29
28	29	30	31	32	1

(d) Permutation Function (P)

16	7	20	21	29	12	28	17
1	15	23	26	5	18	31	10
2	8	24	14	32	27	3	9
19	13	30	6	22	11	4	25

Definición de DES S-Boxes

S_1

14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13

S_2

15	1	8	14	6	11	3	4	9	7	2	13	12	0	5	10
3	13	4	7	15	2	8	14	12	0	1	10	6	9	11	5
0	14	7	11	10	4	13	1	5	8	12	6	9	3	2	15
13	8	10	1	3	15	4	2	11	6	7	12	0	5	14	9

S_3

10	0	9	14	6	3	15	5	1	13	12	7	11	4	2	8
13	7	0	9	3	4	6	10	2	8	5	14	12	11	15	1
13	6	4	9	8	15	3	0	11	1	2	12	5	10	14	7
1	10	13	0	6	9	8	7	4	15	14	3	11	5	2	12

S_4

7	13	14	3	0	6	9	10	1	2	8	5	11	12	4	15
13	8	11	5	6	15	0	3	4	7	2	12	1	10	14	9
10	6	9	0	12	11	7	13	15	1	3	14	5	2	8	4
3	15	0	6	10	1	13	8	9	4	5	11	12	7	2	14

S_5

2	12	4	1	7	10	11	6	8	5	3	15	13	0	14	9
14	11	2	12	4	7	13	1	5	0	15	10	3	9	8	6
4	2	1	11	10	13	7	8	15	9	12	5	6	3	0	14
11	8	12	7	1	14	2	13	6	15	0	9	10	4	5	3

S_6

12	1	10	15	9	2	6	8	0	13	3	4	14	7	5	11
10	15	4	2	7	12	9	5	6	1	13	14	0	11	3	8
9	14	15	5	2	8	12	3	7	0	4	10	1	13	11	6
4	3	2	12	9	5	15	10	11	14	1	7	6	0	8	13

S_7

4	11	2	14	15	0	8	13	3	12	9	7	5	10	6	1
13	0	11	7	4	9	1	10	14	3	5	12	2	15	8	6
1	4	11	13	12	3	7	14	10	15	6	8	0	5	9	2
6	11	13	8	1	4	10	7	9	5	0	15	14	2	3	12

S_8

13	2	8	4	6	15	11	1	10	9	3	14	5	0	12	7
1	15	13	8	10	3	7	4	12	5	6	11	0	14	9	2
7	11	4	1	9	12	14	2	0	6	10	13	15	3	5	8
2	1	14	7	4	10	8	13	15	12	9	0	3	5	6	11

Tablas para armado de llave DES

(b) Permuted Choice One (PC-1)

57	49	41	33	25	17	9
1	58	50	42	34	26	18
10	2	59	51	43	35	27
19	11	3	60	52	44	36
63	55	47	39	31	23	15
7	62	54	46	38	30	22
14	6	61	53	45	37	29
21	13	5	28	20	12	4

(c) Permuted Choice Two (PC-2)

14	17	11	24	1	5	3	28
15	6	21	10	23	19	12	4
26	8	16	7	27	20	13	2
41	52	31	37	47	55	30	40
51	45	33	48	44	49	39	56
34	53	46	42	50	36	29	32

AES S-Boxes

		y														
		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
x	0	63	7C	77	7B	F2	6B	6F	C5	30	01	67	2B	FE	D7	AB	76
	1	CA	82	C9	7D	FA	59	47	F0	AD	D4	A2	AF	9C	A4	72	C0
	2	B7	FD	93	26	36	3F	F7	CC	34	A5	E5	F1	71	D8	31	15
	3	04	C7	23	C3	18	96	05	9A	07	12	80	E2	EB	27	B2	75
	4	09	83	2C	1A	1B	6E	5A	A0	52	3B	D6	B3	29	E3	2F	84
	5	53	D1	00	ED	20	FC	B1	5B	6A	CB	BE	39	4A	4C	58	CF
	6	D0	EF	AA	FB	43	4D	33	85	45	F9	02	7F	50	3C	9F	A8
	7	51	A3	40	8F	92	9D	38	F5	BC	B6	DA	21	10	FF	F3	D2
	8	CD	0C	13	EC	5F	97	44	17	C4	A7	7E	3D	64	5D	19	73
	9	60	81	4F	DC	22	2A	90	88	46	EE	B8	14	DE	5E	0B	DB
	A	E0	32	3A	0A	49	06	24	5C	C2	D3	AC	62	91	95	E4	79
	B	E7	C8	37	6D	8D	D5	4E	A9	6C	56	F4	EA	65	7A	AE	08
	C	BA	78	25	2E	1C	A6	B4	C6	E8	DD	74	1F	4B	BD	8B	8A
	D	70	3E	B5	66	48	03	F6	0E	61	35	57	B9	86	C1	1D	9E
	E	E1	F8	98	11	69	D9	8E	94	9B	1E	87	E9	CE	55	28	DF
	F	8C	A1	89	0D	BF	E6	42	68	41	99	2D	0F	B0	54	BB	16

(a) S-box

		y															
		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
x	0	52	09	6A	D5	30	36	A5	38	BF	40	A3	9E	81	F3	D7	FB
	1	7C	E3	39	82	9B	2F	FF	87	34	8E	43	44	C4	DE	E9	CB
	2	54	7B	94	32	A6	C2	23	3D	EE	4C	95	0B	42	FA	C3	4E
	3	08	2E	A1	66	28	D9	24	B2	76	5B	A2	49	6D	8B	D1	25
	4	72	F8	F6	64	86	68	98	16	D4	A4	5C	CC	5D	65	B6	92
	5	6C	70	48	50	FD	ED	B9	DA	5E	15	46	57	A7	8D	9D	84
	6	90	D8	AB	00	8C	BC	D3	0A	F7	E4	58	05	B8	B3	45	06
	7	D0	2C	1E	8F	CA	3F	0F	02	C1	AF	BD	03	01	13	8A	6B
	8	3A	91	11	41	4F	67	DC	EA	97	F2	CF	CE	F0	B4	E6	73
	9	96	AC	74	22	E7	AD	35	85	E2	F9	37	E8	1C	75	DF	6E
	A	47	F1	1A	71	1D	29	C5	89	6F	B7	62	0E	AA	18	BE	1B
	B	FC	56	3E	4B	C6	D2	79	20	9A	DB	C0	FE	78	CD	5A	F4
	C	1F	DD	A8	33	88	07	C7	31	B1	12	10	59	27	80	EC	5F
	D	60	51	7F	A9	19	B5	4A	0D	2D	E5	7A	9F	93	C9	9C	EF
	E	A0	E0	3B	4D	AE	2A	F5	B0	C8	EB	BB	3C	83	53	99	61
	F	17	2B	04	7E	BA	77	D6	26	E1	69	14	63	55	21	0C	7D

(b) Inverse S-box

Referencias

[1] Jeffrey Hoffstein, Jill Pipher y Joseph H. Silverman. An Introduction to Mathematical Cryptography. Springer Science + Business Media. First Edition, 2008.
[2] Jonathan Katz and Yehuda Lindell. Introduction to Modern Cryptography. Chapman and Hall/CRC. First Edition, 2007.
[3] William Stallings. Cryptography and Network Security. Pearson Prentice Hall. Fifth Edition, 2011. Principles and Practice.
[4] Martin Fowler. Patterns of Enterprise Application Architecture. Pearson Education, First Edition, 2003.
[5] Andrew S. Tanenbaum. Modern Operating Systems. Pearson Prentice Hall. Third Edition, 2009.
[6] Jan Axelson. USB Mass Storage. Lakeview Research LLC Madison, WI. First Edition, 2006. Designing and Programming Devices and Embedded Hosts.
[7] Criptografía. Definición de criptografía. Consulta febrero 2015. http://es.wikipedia.org/wiki/Criptograf%C3%ADa
[8] Wolfram MathWorld. Greatest Common Divisor. Consulta febrero 2015. http://mathworld.wolfram.com/GreatestCommonDivisor.html
[9] David Austin. AMS American Mathematical Society. Random Numbers: Nothing Left to Chance. Consulta febrero 2015. http://www.ams.org/samplings/feature-column/fcarc-random
[10] Randomness Recommendations for Security. IETF publicación RFC1750, Internet Engineering Task Force, diciembre 1994. http://www.ietf.org/rfc/rfc1750.txt
[11] Cryptographic Hash and SHA-3 Standard Development. NIST publicación SHA-3, National Institute of Standards and Technology, marzo 2014. http://csrc.nist.gov/groups/ST/hash/sha-3/
[12] Compilador. Definición de compilador. Consulta febrero 2015. http://es.wikipedia.org/wiki/Compilador
[13] Programación Orientada a Objetos. Conceptos fundamentales. Consulta febrero 2015. http://es.wikipedia.org/wiki/Programaci%C3%B3n_orientada_a_objetos
[14] ¿Qué es un Patrón de Diseño?. Consulta febrero 2015. http://msdn.microsoft.com/es-es/library/bb972240.aspx
[15] Framework. Definición de Framework. Consulta febrero 2015. http://es.wikipedia.org/wiki/Framework
[16] Spring Framework Reference. Modules. Consulta febrero 2015. http://docs.spring.io/spring/docs/current/spring-framework-reference/htmlsingle
[17] Model-View-Controller. Consulta febrero 2015. http://msdn.microsoft.com/en-us/library/ff649643.aspx

[18] Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing. IETF publicación RFC 7230, Internet Engineering Task Force, junio 2014. https://tools.ietf.org/html/rfc7230
[19] Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content. IETF publicación RFC7231, Internet Engineering Task Force, junio 2014. https://tools.ietf.org/html/rfc7231
[20] Tiles. Definición de Tiles. Consulta febrero 2015. https://tiles.apache.org/
[21] AngularJS. Definición de AngularJS. Consulta febrero 2015. https://angularjs.org/
[22] Java Documentation. Invoking JavaScript Code from Applet. Consulta febrero 2015. https://docs.oracle.com/javase/tutorial/deployment/applet/invokingJavaScriptFromApplet.html
[23] The Linux Kernel Archives. About Linux Kernel. Publicación agosto 2013. https://www.kernel.org/linux.html
[24] The Linux Kernel Archives. Man udev. Consulta febrero 2015. https://www.kernel.org/pub/linux/utils/kernel/hotplug/udev/udev.html
[25] The Open Group. What is UNIX? Consulta febrero 2015. http://www.unix.org/what_is_unix.html
[26] Microsoft Windows. A history of Windows. Publicación noviembre 2013. http://windows.microsoft.com/en-us/windows/history#T1=era0
[27] Microsoft TechNet. How Plug and Play Works. Publicación marzo 2003. http://technet.microsoft.com/en-us/library/cc781092%28v=ws.10%29.aspx
[28] Microsoft MSDN. USB device descriptors. Consulta marzo 2015. http://msdn.microsoft.com/en-us/library/windows/hardware/ff539283%28v=vs.85%29.aspx
[29] Libusb. API documentation. Publicación febrero 2015. http://libusb.org/
[30] A Layman's Guide to a Subset of ASN.1, BER, and DER. Publicación noviembre 1993. http://luca.ntop.org/Teaching/Appunti/asn1.html
[31] ARM mbed. ASN.1 key structures in DER and PEM. Publicación abril 2014. https://tls.mbed.org/kb/cryptography/asn1-key-structures-in-der-and-pem
[32] Public-Key Cryptography Standards (PKCS) #8: Private-Key Information Syntax Specification Version 1.2. IETF publicación RFC 5208, Internet Engineering Task Force, mayo 2008. http://www.ietf.org/rfc/rfc5208.txt
[33] Recommendation for Block Cipher Modes of Operation, NIST publicación 800-38A, National Institute of Standards and Technology, diciembre 2001. http://csrc.nist.gov/publications/nistpubs/800-38a/sp800-38a.pdf
[34] PKCS #12: Personal Information Exchange Syntax v1.1. IETF publicación RFC 7292, Internet Engineering Task Force, julio 2014. https://tools.ietf.org/html/rfc7292
[35] OpenSSL. OpenSSL Documents. Consulta febrero 2015. https://www.openssl.org/docs/
[36] Mac Developer Library. System_profiler. Publicación junio 2003. https://developer.apple.com/library/mac/documentation/Darwin/Reference/ManPages/man8/system_profiler.8.html

[37] Beyond Logic. USB Descriptors. Publicación septiembre 2010. http://www.beyondlogic.org/usbnutshell/usb5.shtml
[38] Java Documentation. Java Native Interface Specification. Consulta Febrero 2014. http://docs.oracle.com/javase/6/docs/technotes/guides/jni/spec/jniTOC.html
[39] Security Token. Physical types. Publicación enero 2015. http://en.wikipedia.org/wiki/Security_token
[40] Federal Ministry of the Interior. German National Identity Card. Consulta febrero 2015. http://www.personalausweisportal.de/EN/Home/home_node.html
[41] Java Documentation. JAR File Manifest Attributes for Security. Consulta febrero 2015. http://docs.oracle.com/javase/7/docs/technotes/guides/jweb/security/manifest.html
[42] Java Documentation. Signing Applets Using RSA Certificates. Consulta febrero 2015. http://docs.oracle.com/javase/7/docs/technotes/guides/jweb/security/rsa_signing.html
[43] Java Documentation. Jarsigner. Consulta febrero 2015. http://docs.oracle.com/javase/7/docs/technotes/tools/windows/jarsigner.html